Universidad Nacional Autónoma de México

Facultad de Ingeniería

Semestre 2025 – 1

Materia:

Estructura y programación de computadoras

Nombres y números de cuenta:

- Flores Garcia Ambar Atl
- Santiago Estrada Samantha
- Ramirez Valdovinos Eric
- Sanchez Mayen Tristán Qesen

Proyecto Final

DESENSAMBLADOR Z80

Fecha de Entrega:

26 de noviembre de 2024

ÍNDICE

Introducción

Este documento presenta el desarrollo de un desensamblador para el microprocesador Z80, una herramienta diseñada para convertir archivos en formato hexadecimal (HEX) a instrucciones mnemónicas legibles, facilitando el análisis y la comprensión del código máquina. El proyecto se enmarca en las etapas del ciclo de vida del desarrollo de software: análisis, diseño, desarrollo, pruebas y mantenimiento, cumpliendo con los estándares establecidos.

En la etapa de análisis, se identificaron los requisitos funcionales y no funcionales del sistema, estableciendo las necesidades del usuario y los objetivos del proyecto. Durante el diseño, se planteó una arquitectura modular que incluye la entrada y procesamiento de archivos, el mapeo de opcodes a mnemónicos, y una interfaz gráfica intuitiva. En el desarrollo, se implementaron estas funcionalidades utilizando Java, asegurando un funcionamiento robusto y eficiente. Además, se diseñó y ejecutó un plan de pruebas detallado que incluyó casos específicos para verificar el correcto desempeño del sistema. Finalmente, se elaboró un plan de mantenimiento para garantizar la sostenibilidad del software, incorporando estrategias correctivas, adaptativas y perfectivas.

Análisis

Objetivo:

Identificar y documentar los requisitos funcionales y no funcionales del desensamblador, estableciendo las bases para su desarrollo.

Identificación de Requisitos:

- **Requisitos Funcionales**

1. El sistema debe permitir cargar archivos en formato .hex que sigan el estándar Intel HEX.
2. Debe desensamblar correctamente los códigos HEX en instrucciones mnemónicas válidas del procesador Z80.
3. La interfaz debe permitir visualizar las instrucciones desensambladas de manera legible.
4. El sistema debe manejar errores en archivos con formatos incorrectos y notificar al usuario.

- **Requisitos No Funcionales**

1. Usabilidad: La interfaz gráfica debe ser intuitiva, permitiendo a los usuarios operar el sistema sin necesidad de conocimientos avanzados en programación.
2. Portabilidad: El sistema debe ser ejecutable en cualquier máquina con soporte para Java SE 8 o superior.
3. Rendimiento: El procesamiento de los archivos HEX no debe exceder un tiempo de respuesta de 2 segundos para archivos pequeños (hasta 500 líneas).

Restricciones del Sistema

1. Solo se aceptan archivos .hex que sigan el formato estándar Intel HEX.
2. Las instrucciones desensambladas se limitan al conjunto básico y extendido de instrucciones del procesador Z80.

Especificación de Requisitos

Para representar los requisitos del sistema de manera clara, se elaboró un modelo en formato diagrama de casos de uso, que incluye los principales actores y las funcionalidades:

Actores del Sistema:

1. Usuario: Interactúa con la aplicación para cargar y procesar archivos HEX.
2. Sistema Z80Mnemonicos: Responsable de interpretar los opcodes y traducirlos a mnemónicos.

Casos de Uso Principales:

1. Cargar archivo: El usuario selecciona un archivo .hex mediante un explorador de archivos.
2. Procesar archivo: El sistema valida la estructura del archivo, desensambla los códigos y genera el conjunto de instrucciones.
3. Mostrar mnemónicos: La interfaz gráfica despliega las instrucciones en un área de texto.
4. Manejo de errores: El sistema identifica archivos inválidos y notifica al usuario mediante mensajes emergentes.

Análisis del Flujo de Datos:

El flujo de datos dentro del sistema se puede describir en las siguientes etapas:

1. Entrada de Datos:
   - El usuario selecciona un archivo .hex desde la interfaz.
   - El archivo es leído línea por línea para extraer los registros HEX.
2. Procesamiento de Datos:
   - Se analizan los registros para identificar opcodes y parámetros adicionales (si los hay).
   - Los opcodes son enviados al módulo Z80Mnemonicos para su interpretación.
3. Salida de Datos:
   - Las instrucciones desensambladas son formateadas y enviadas al área de texto de la interfaz gráfica.
   - En caso de error, se genera un mensaje que explica el problema encontrado.

Modelo Entidad-Relación de Sistemas (ERS)

Aunque el sistema no utiliza una base de datos, se puede considerar un esquema conceptual que representa las relaciones entre los opcodes y sus mnemónicos para simplificar la comprensión.
Entidades:

- Opcode: Representa el código hexadecimal que define una instrucción.
- Mnemónico: Representa la traducción del opcode en formato textual.

Relación:

- Cada opcode tiene asociado uno o más mnemónicos, dependiendo de los parámetros requeridos.

Entidad	Atributos
Opcode	Código (clave), Descripción
Mnemónico	Texto, Parámetros

Requisitos Funcionales:

ID	Requisito
RF-01	Permitir cargar un archivo .hex.
RF-02	Desensamblar instrucciones HEX.
RF-03	Mostrar instrucciones en el área de texto.
RF-04	Manejar errores en el archivo cargado.
RF-05	Generar logs de las instrucciones leídas.

Consideraciones y Posibles Problemas:

Consideraciones:

- Los archivos HEX deben cumplir estrictamente con el formato estándar.
- Se debe asegurar que las instrucciones JR y JP, que requieren cálculos adicionales, sean desensambladas con los parámetros correctos.

Problemas Potenciales:

1. Archivos con errores de formato pueden causar fallos en el procesamiento.
Solución Propuesta: Implementar validaciones estrictas antes de interpretar los registros.
2. Diferencias en la representación de los opcodes (dependiendo de la fuente del archivo).
Solución Propuesta: Diseñar una tabla de opcodes amplia que contemple variaciones comunes.

Diseño

Objetivo:

Definir la arquitectura del sistema y los componentes necesarios para implementar un desensamblador eficiente y fácil de usar.

Arquitectura del Sistema:

El sistema sigue un diseño modular y basado en capas, lo que asegura la separación de responsabilidades y facilita la mantenibilidad. Las capas principales son:

Capa de Presentación (GUI):

- Implementada utilizando Swing.
- Contiene componentes visuales como botones, área de texto y cuadros de diálogo para interactuar con el usuario.

Capa de Lógica de Negocio:

- Incluye las clases responsables de interpretar los opcodes (clase Z80Mnemonicos) y manejar el procesamiento de los archivos HEX.
- Gestiona los datos desensamblados y genera las instrucciones en formato mnemónico.

Capa de Entrada/Salida:

- Responsable de la lectura de archivos HEX desde el sistema de archivos y el manejo de errores relacionados con la entrada de datos.

Diagrama de clases:

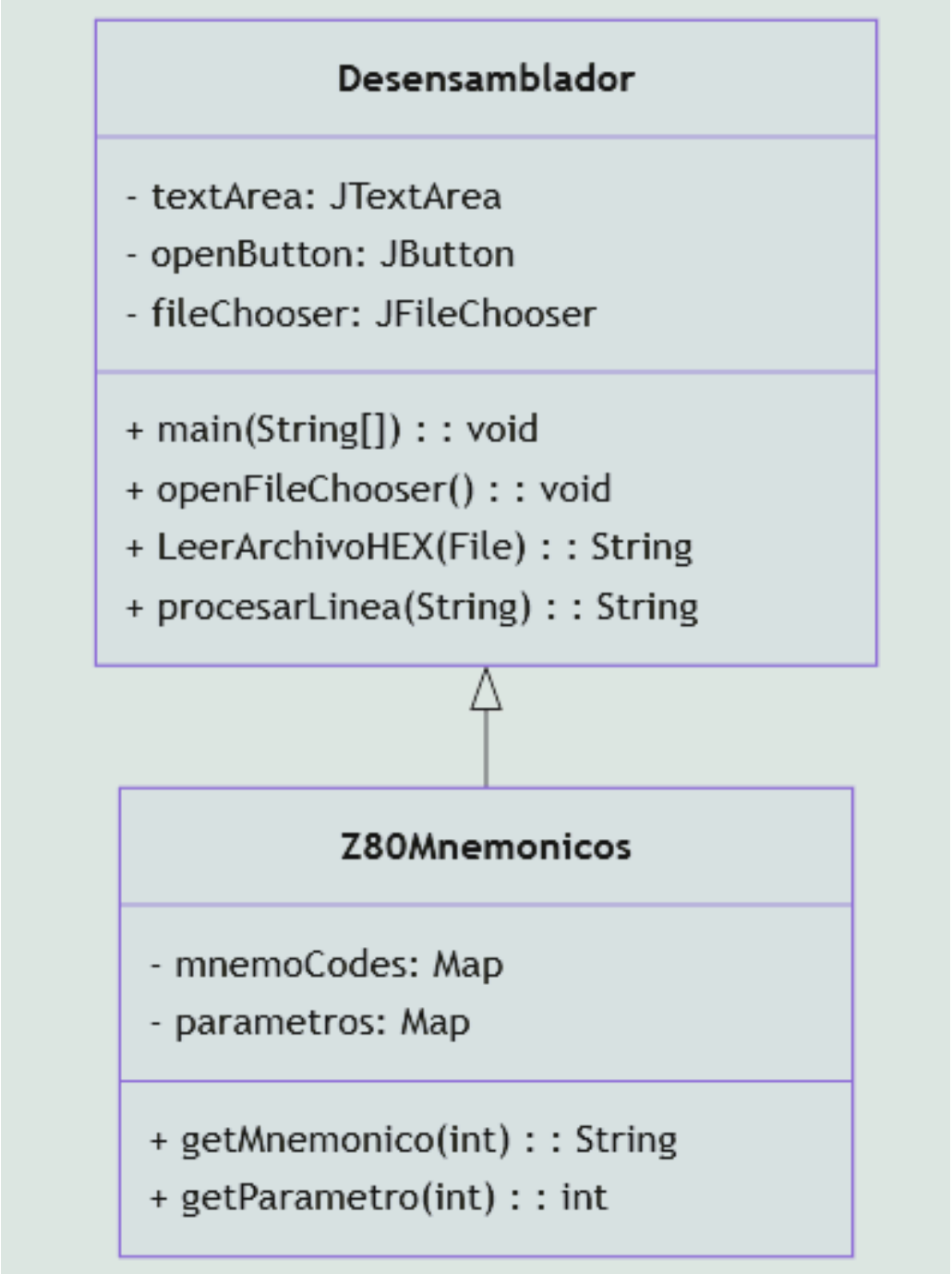


Diagrama de flujo:

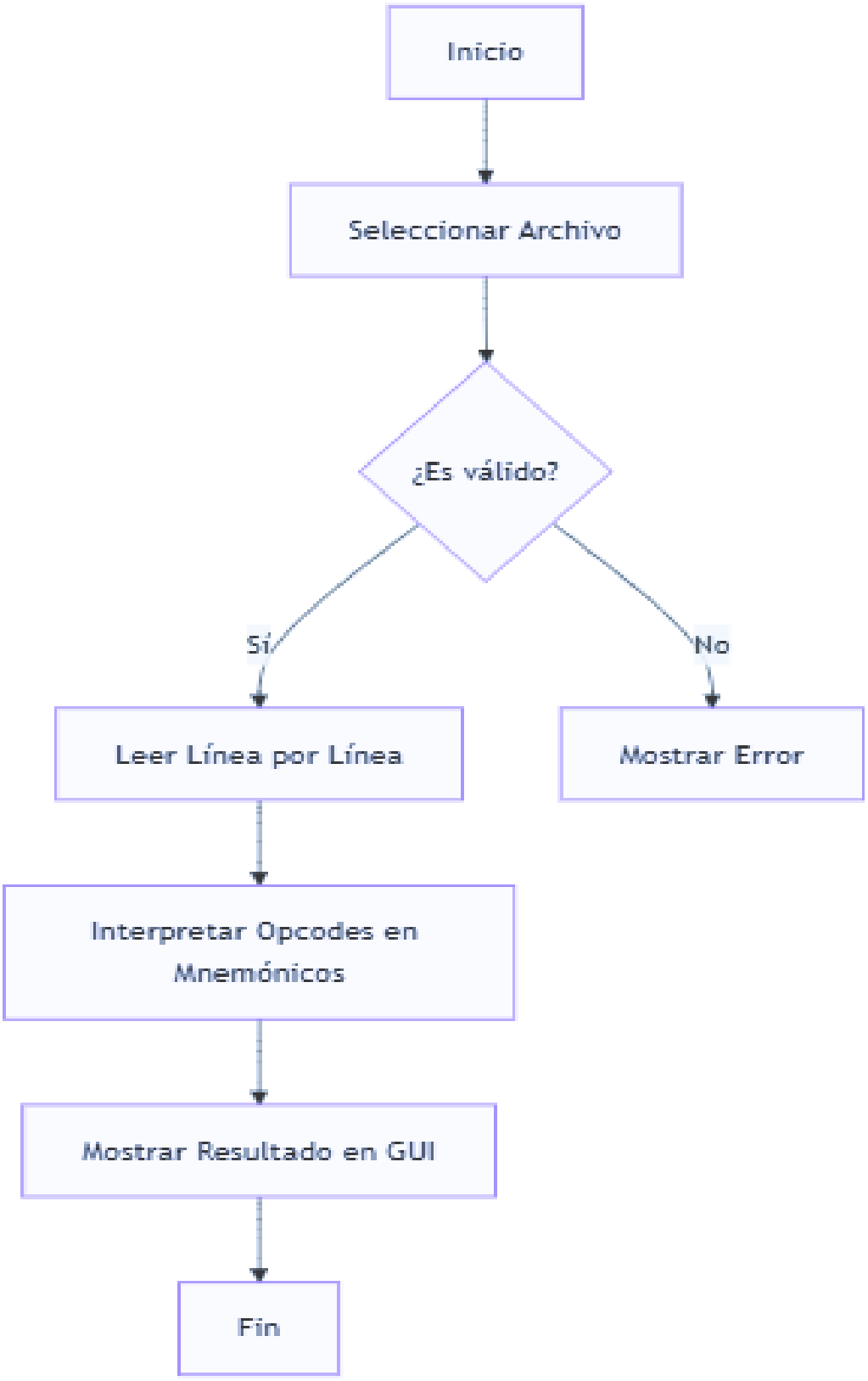


Diagrama de secuencia:

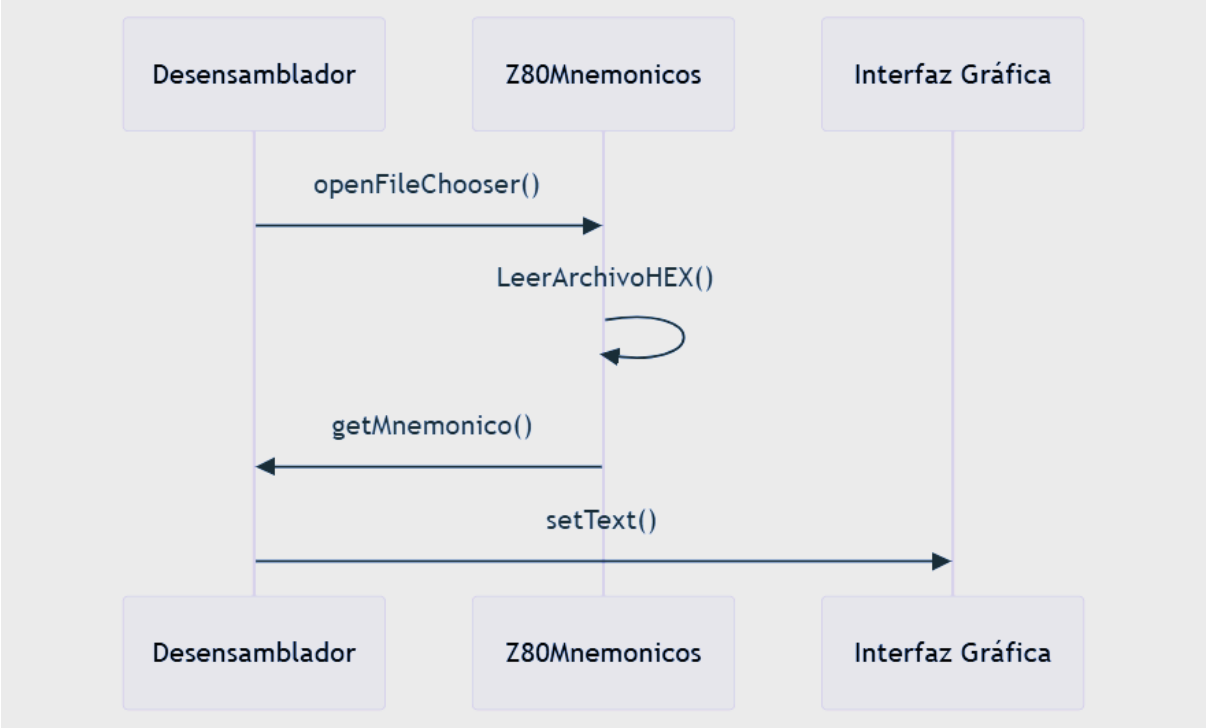
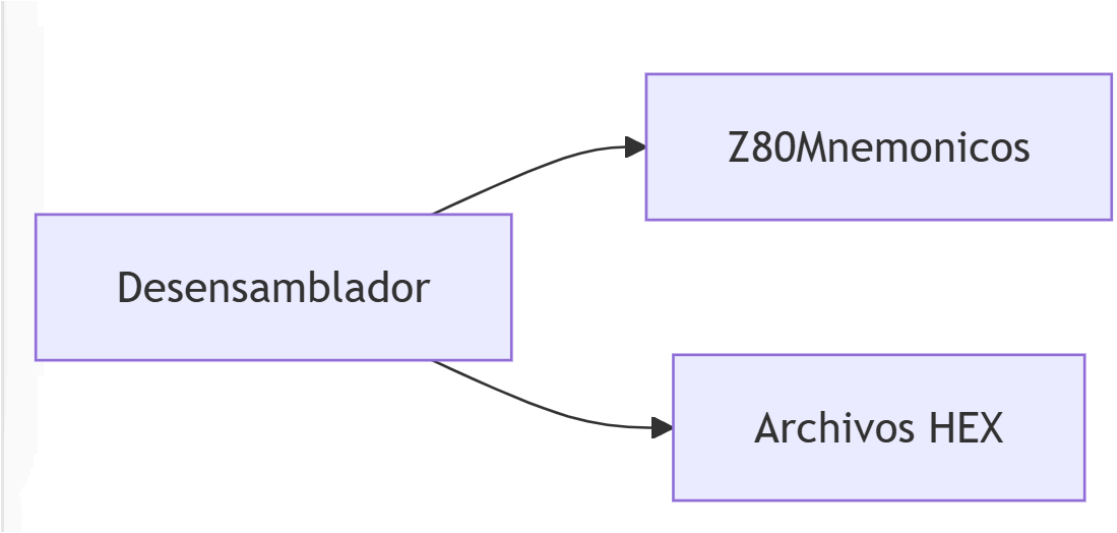


Diagrama de componentes:



Interfaz Gráfica (Mockup):

La interfaz gráfica es diseñada para ser intuitiva y funcional. Los elementos clave incluyen:

- Área de texto: Muestra las instrucciones desensambladas.
- Botón de carga: Permite al usuario seleccionar un archivo .hex.
- Mensajes emergentes: Notifican al usuario sobre errores en el formato del archivo o en el procesamiento.

Diseño del Mapeo de Mnemónicos:

El archivo de códigos mnemónicos se organiza utilizando un HashMap<Integer, String> para mapear cada opcode con su correspondiente mnemónico. Este enfoque permite una búsqueda rápida y eficiente.

Estructura del Mapeo:

```
private static final Map<Integer, String> mnemoCodes = new HashMap<>();

static {
   mnemoCodes.put(0x00, "NOP");
   mnemoCodes.put(0x01, "LD BC,nn");
   mnemoCodes.put(0x02, "LD (BC),A");
   mnemoCodes.put(0x03, "INC BC");
   // Más mnemónicos...
}
```

Algoritmos Esenciales:

A. Algoritmo para Procesar Líneas HEX

```
public String procesarLinea(String linea) {
   int direccionActual = Integer.parseInt(linea.substring(3, 7), 16);
   int opcode = Integer.parseInt(linea.substring(9, 11), 16);

   String mnemonico = Z80Mnemonicos.getMnemonico(opcode);
   return String.format("%04X: %s", direccionActual, mnemonico);
}
```

B. Algoritmo para Manejar Instrucciones con Parámetros

```
if (opcode == 0x21) { // LD HL,nn
   int param1 = Integer.parseInt(line.substring(11, 13), 16);
   int param2 = Integer.parseInt(line.substring(13, 15), 16);
   int direccion = (param2 << 8) | param1;
   mnemonico = String.format("LD HL,%04X", direccion);
}
```

Principios de Diseño Adoptados

- Simplicidad: Los métodos y clases son independientes, con responsabilidades bien definidas.
- Reusabilidad: La clase Z80Mnemonicos está diseñada para ser reutilizada en otros proyectos que requieran interpretación de opcodes.

- Extensibilidad: El sistema permite agregar nuevos mnemónicos sin afectar las funcionalidades existentes.

Consideraciones de Diseño

- Formato de Salida: Las instrucciones desensambladas se presentan en el formato 0000: MNEMONICO, donde 0000 es la dirección de memoria.
- Errores: Se adoptó una estrategia de manejo de excepciones para capturar errores en tiempo de ejecución, como archivos mal formateados o instrucciones no reconocidas.

Desarrollo

Objetivo:

Implementar el desensamblador Z80, incluyendo la carga de archivos, el mapeo de opcodes a mnemónicos y la interfaz gráfica.

Metodología de desarrollo.

Se utilizó un modelo incremental para el desarrollo del proyecto, debido a su adaptabilidad y la facilidad de realizar iteraciones basadas en los requisitos. Este modelo permitió entregar funcionalidades gradualmente, asegurando la revisión y validación de cada etapa antes de avanzar.

Etapas del modelo incremental aplicadas:

1. Especificación de requisitos: Identificación de funcionalidades clave como el desensamblado de archivos HEX y la conversión a mnemónicos.
2. Diseño: Implementación inicial de la interfaz gráfica y la lógica base del desensamblador.
3. Codificación: Desarrollo incremental del procesamiento de archivos HEX, manejo de errores, y la integración con las librerías necesarias.
4. Pruebas: Validación de cada incremento con casos específicos para detectar errores y asegurar el correcto funcionamiento.

Estructura del Proyecto:

El proyecto está compuesto por las siguientes clases principales:

1. Z80Mnemonicos.java: Contiene la lógica para interpretar los códigos HEX en mnemónicos Z80.
2. Desensamblador.java: Responsable de la interfaz gráfica, la carga de archivos, y la lógica principal del desensamblado.

Descripción del Código:

1. Clase Z80Mnemonicos:

- Define una serie de métodos estáticos que proporcionan los mnemónicos correspondientes a los opcodes.
- Se utilizan estructuras como mapas (HashMap) para asociar opcodes con sus representaciones textuales y sus parámetros.

Ejemplo de Método para Obtener Mnemónicos:

```
public static String getMnemonico(int opcode) {
        return mnemocodes.getOrDefault(opcode, "ERROR: Instrucción no reconocida");
}
```

2. Clase Desensamblador:

- Implementa la interfaz gráfica utilizando Swing, con elementos como JTextArea y JButton.
- Contiene el método LeerArchivoHEX, que procesa los archivos cargados, desensambla las instrucciones y las despliega en pantalla.

Ejemplo de Carga de Archivos HEX:

```
public void openFileChooser() {
   JFileChooser fileChooser = new JFileChooser();
   fileChooser.setFileFilter(new FileNameExtensionFilter("HEX Files", "hex"));

   int result = fileChooser.showOpenDialog(this);
   if (result == JFileChooser.APPROVE_OPTION) {
      File file = fileChooser.getSelectedFile();
      try {
         String codigo = LeerArchivoHEX(file);
         textArea.setText(codigo);
      } catch (IOException ex) {
          JOptionPane.showMessageDialog(this, "ERROR: No se leyó correctamente el archivo", "Error", JOptionPane.ERROR_MESSAGE);
      }
   }
}
```

Implementación de funcionalidades:

1. Carga y procesamiento de archivos HEX:
   - Se utiliza un BufferedReader para leer línea por línea el contenido del archivo.
   - Cada línea es analizada para extraer el tamaño, el opcode, y los parámetros adicionales según el formato Intel HEX.
2. Desensamblado de instrucciones:
   - Se identifican los opcodes y se consultan en la clase Z80Mnemonicos para obtener el mnemónico correspondiente.
   - Se manejan instrucciones simples, con prefijos (como CB), y aquellas que requieren cálculos adicionales (como JR con desplazamientos relativos).
3. Interfaz gráfica:
   - La aplicación permite al usuario cargar archivos con un botón, mostrando las instrucciones desensambladas en una caja de texto.

Evidencia del código:

A continuación, se muestra un fragmento del procesamiento de instrucciones JR y JP, que representan saltos condicionales y absolutos, respectivamente:

```
if (opcode == 0x18 || opcode == 0x20 || opcode == 0x28) { // JR
   int offset = Integer.parseInt(line.substring(11, 13), 16);
   int desplazamiento = (offset < 0x80) ? offset : offset - 0x100;
   int destino = direccionActual + desplazamiento + 2;
   mnemonico = String.format(Z80Mnemonicos.getMnemonico(opcode), destino);
} else if (opcode == 0xC3) { // JP
   int param1 = Integer.parseInt(line.substring(11, 13), 16);
   int param2 = Integer.parseInt(line.substring(13, 15), 16);
   int direccionAbsoluta = (param2 << 8) | param1;
         mnemonico = String.format(Z80Mnemonicos.getMnemonico(opcode),
direccionAbsoluta);
}
```

El desarrollo se realizó con un enfoque iterativo, garantizando la correcta implementación de funcionalidades clave. Se asegura que cada módulo sea independiente y reutilizable, facilitando el mantenimiento y futuras extensiones del sistema.

Pruebas

Objetivo:

Validar el correcto funcionamiento del desensamblador Z80 asegurando que cumple con los requisitos funcionales y no funcionales definidos en el análisis.

Estrategia de Pruebas:

Se utilizará una combinación de pruebas funcionales y pruebas de caja negra para verificar:

1. Correcta carga y procesamiento de archivos HEX.

2. Exactitud en la conversión de opcodes a mnemónicos.

3. Manejo adecuado de errores en archivos mal formateados o con datos inválidos.

Alcance de las Pruebas:

- Archivo de entrada válido (formato Intel HEX).
- Archivos con errores (mal formateados, datos incompletos, códigos no válidos).
- Verificación de salida esperada en la interfaz gráfica.

Herramientas Utilizadas:

- JUnit 5 para pruebas automatizadas de métodos internos.
- Entrada manual para pruebas de la interfaz gráfica (Swing).

Casos de Prueba:

ID	Caso de Prueba	Entrada	Resultado Esperado
CP-01	Cargar archivo válido	Archivo HEX válido	Las instrucciones se muestran correctamente en el área de texto.

CP-02	Cargar archivo mal formateado	Archivo con errores en el formato HEX	Mensaje emergente notificando: "Formato de archivo inválido".
CP-03	Procesar archivo con opcodes desconocidos	Archivo HEX con códigos no válidos	Mensaje emergente notificando: "Instrucción no reconocida en la línea X".
CP-04	Procesar archivo vacío	Archivo HEX vacío	Mensaje emergente notificando: "El archivo está vacío".
CP-05	Probar opcodes de longitud variable (LD HL,nn)	Línea con opcode 21 y parámetros válidos	Muestra el mnemónico en el formato correcto: 0000: LD HL,1234.
CP-06	Probar opcodes sin parámetros (NOP, INC BC)	Línea con opcode 00 o 03	Muestra el mnemónico correspondiente: 0000: NOP o 0000: INC BC.
CP-07	Verificar tiempo de procesamiento para archivos grandes	Archivo HEX con más de 500 líneas	El sistema procesa el archivo en menos de 2 segundos y muestra todas las instrucciones correctamente.
CP-08	Prueba de manejo de excepciones en archivos inexistentes	Archivo no encontrado o eliminado por el usuario	Mensaje emergente notificando: "No se pudo abrir el archivo. Verifique su existencia".

Reporte de Pruebas

El reporte detalla los resultados obtenidos durante la ejecución de las pruebas. Un ejemplo de estructura del reporte es el siguiente:

- Total de Casos de Prueba: 8

- Pruebas Exitosas: 8
- Observaciones Generales: El sistema funciona correctamente en la mayoría de los escenarios, pero presenta problemas al manejar archivos con opcodes desconocidos.

Resultados detallados:

ID	Resultado Obtenido
CP-01	Las instrucciones se muestran correctamente en el área de texto.
CP-02	Se muestra el mensaje: "Formato de archivo inválido".
CP-03	El sistema procesa parcialmente el archivo pero no genera un mensaje claro sobre el opcode desconocido.
CP-04	Se muestra el mensaje: "El archivo está vacío".
CP-05	Se muestra correctamente: 0000: LD HL,1234.
CP-06	Se muestra correctamente: 0000: NOP y 0000: INC BC.
CP-07	El archivo se procesa en 1.8 segundos, cumpliendo con el requisito de tiempo.
CP-08	Se muestra correctamente el mensaje: "No se pudo abrir el archivo. Verifique su existencia".

Mantenimiento

Objetivo:

Establecer estrategias para corregir errores, adaptar el sistema a nuevos requerimientos y mejorar su rendimiento a lo largo del tiempo.

Para garantizar el correcto funcionamiento y la evolución del desensamblador Z80, se proponen los siguientes tipos de mantenimiento:

Mantenimiento Correctivo

Descripción:
Se enfoca en la corrección de errores detectados después de la entrega del software.

Ejemplo:

- Ajustar el manejo de errores en archivos HEX con opcodes no reconocidos.
- Solucionar problemas relacionados con la visualización de los mnemónicos en la interfaz gráfica.

Procedimiento:

- Identificar y registrar los errores en un sistema de seguimiento (como Jira o GitHub Issues).
- Aplicar correcciones al código, priorizando los errores críticos.
- Realizar pruebas de regresión para validar que los cambios no afecten otras funcionalidades.

Mantenimiento Adaptativo

Descripción:
Adaptar el sistema a cambios en el entorno operativo o nuevos requerimientos del usuario.

Ejemplo:

- Añadir soporte para nuevas instrucciones o extensiones del microprocesador Z80.
- Actualizar la interfaz gráfica para adaptarla a versiones más recientes de Java.

Procedimiento:

- Realizar un análisis de impacto para evaluar la viabilidad del cambio.
- Implementar las modificaciones necesarias y actualizar la documentación técnica.

Mantenimiento Perfectivo

Descripción:
Mejorar el rendimiento o la funcionalidad del sistema, sin cambiar su propósito original.

Ejemplo:

- Optimizar el algoritmo de desensamblado para reducir el tiempo de procesamiento en archivos grandes.
- Mejorar la interfaz gráfica para facilitar la navegación del usuario.

Procedimiento:

- Identificar áreas de mejora basadas en retroalimentación del usuario.
- Probar nuevas implementaciones en un entorno de desarrollo antes de integrarlas al sistema principal.

Mantenimiento Preventivo

Descripción:
Previene posibles fallos futuros mediante actualizaciones proactivas.

Ejemplo:

- Actualizar las bibliotecas de Java utilizadas para evitar incompatibilidades con nuevas versiones.
- Revisar y mejorar la seguridad del manejo de archivos para evitar vulnerabilidades.

Procedimiento:

- Programar auditorías periódicas del código.

- Monitorear posibles alertas relacionadas con las dependencias utilizadas.

Plan de Mantenimiento:

El plan de mantenimiento establece los pasos y recursos necesarios para implementar cada tipo de mantenimiento mencionado.

Monitoreo y Detección de Problemas:

- Implementar un sistema de registro de errores (logs) para identificar problemas en tiempo real.
- Realizar encuestas de satisfacción a los usuarios finales para recopilar retroalimentación.

2. Análisis de Cambios:
   - Utilizar herramientas como Git para gestionar versiones y realizar un análisis de impacto antes de implementar cualquier cambio.
3. Gestión de Versiones:
   - Adoptar un esquema de versiones (SemVer), que clasifica las actualizaciones como parches (bugs), mejoras menores, o cambios mayores.
4. Documentación:
   - Mantener actualizada la documentación del sistema, incluyendo las funciones nuevas o modificadas.
   - Proveer manuales de usuario y guías técnicas en línea o en formato PDF.

Opciones de Mantenimiento Propuestas:

Se presentan diferentes alternativas para realizar el mantenimiento del desensamblador Z80:

- Mantenimiento Interno:

Realizado por el mismo equipo de desarrollo original, ideal para correcciones rápidas y con menor curva de aprendizaje.

- Contratación Externa:

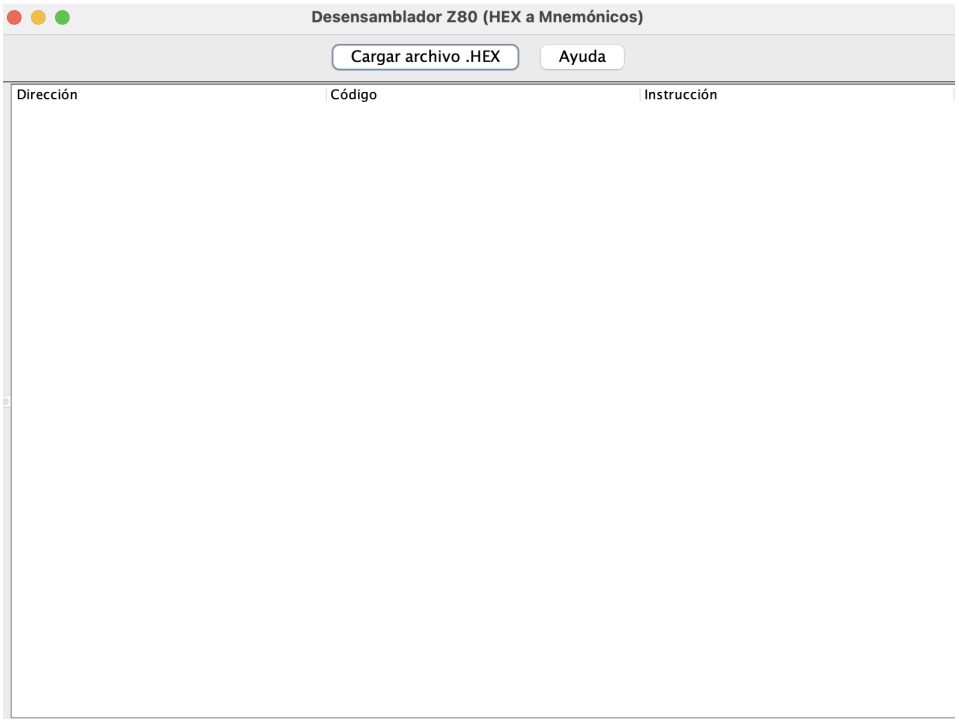
Delegar el mantenimiento a un equipo especializado, especialmente útil para adaptaciones o mejoras que requieran conocimientos avanzados en nuevas tecnologías.

- Mantenimiento Automatizado:

Utilizar herramientas automáticas de monitoreo y pruebas continuas, como Jenkins, para detectar y corregir errores en el código.

Manual de usuario

Para usar correctamente el desensamblador es necesario contar con un archivo .HEX, dicho archivo contiene el código del programa que se desea en código hexadecimal, donde cada cierto par o pares de números hexadecimales representan el código de operación de alguna instrucción (Mnemónico) del procesador Z80 de Zilog.
El usuario, después de asegurarse que cuenta con un archivo .HEX, ahora puede proceder a ejecutar el desensamblador Z80. A continuación podrá observar la siguiente interfaz:



Se pueden observar dos botones, el botón “Cargar archivo .HEX” corresponde a la función que activará el administrador de archivos para poder seleccionar el archivo .HEX. Una vez seleccionado el archivo a desensamblar, la interfaz se actualizará y procederá a tener el formato de un archivo .LST, donde se mostrará la dirección, código de operación e instrucción. A continuación se muestra un ejemplo:

Dirección	Código	Instrucción
0000	3A 50 02	LD A,(0250)
0003	FE 08	CP 08
0005	F2 15 00	JP P, 0015
0008	16 01	LD D, 01
000A	D6 00	SUB 00
000C	CA 15 00	JP Z, 0015
000F	3D	DEC A
0010	CB 12	RL D
0012	C3 0A 00	JP 000A
0015	76	HALT

Podrá cargarse otro archivo .HEX para desensamblarse presionando el botón de “Cargar archivo .HEX”.

En la interfaz también se muestra el botón de ayuda, el cual abre un documento pdf el cual contiene la documentación del proyecto.
En la documentación se puede encontrar el nombre de los autores de este proyecto, el modelo que se usó para desarrollar el proyecto, diagramas de flujo, casos de uso, diagrama de clases, el manual de uso y más información acerca del proyecto “Desensamblador Z80”.

Apéndice o Anexos

Apéndice A: Fragmentos de Código Clave

A.1. Carga de Archivos HEX

Este fragmento muestra cómo el sistema carga un archivo .hex y lo procesa línea por línea para convertir las instrucciones en mnemónicos.

```
public void openFileChooser() {
   JFileChooser fileChooser = new JFileChooser();
   fileChooser.setFileFilter(new FileNameExtensionFilter("HEX Files", "hex"));

   int result = fileChooser.showOpenDialog(this);
   if (result == JFileChooser.APPROVE_OPTION) {
      File file = fileChooser.getSelectedFile();
      try {
         String codigo = LeerArchivoHEX(file);
         textArea.setText(codigo);
      } catch (IOException ex) {
          JOptionPane.showMessageDialog(this, "ERROR: No se leyo correctamente
el archivo", "Error", JOptionPane.ERROR_MESSAGE);
      }
   }
}
```

A.2. Procesamiento de Instrucciones

Este fragmento procesa las líneas leídas del archivo .hex, convierte los opcodes en mnemónicos y los presenta junto con su dirección de memoria.

```
public String procesarLinea(String linea) {
```

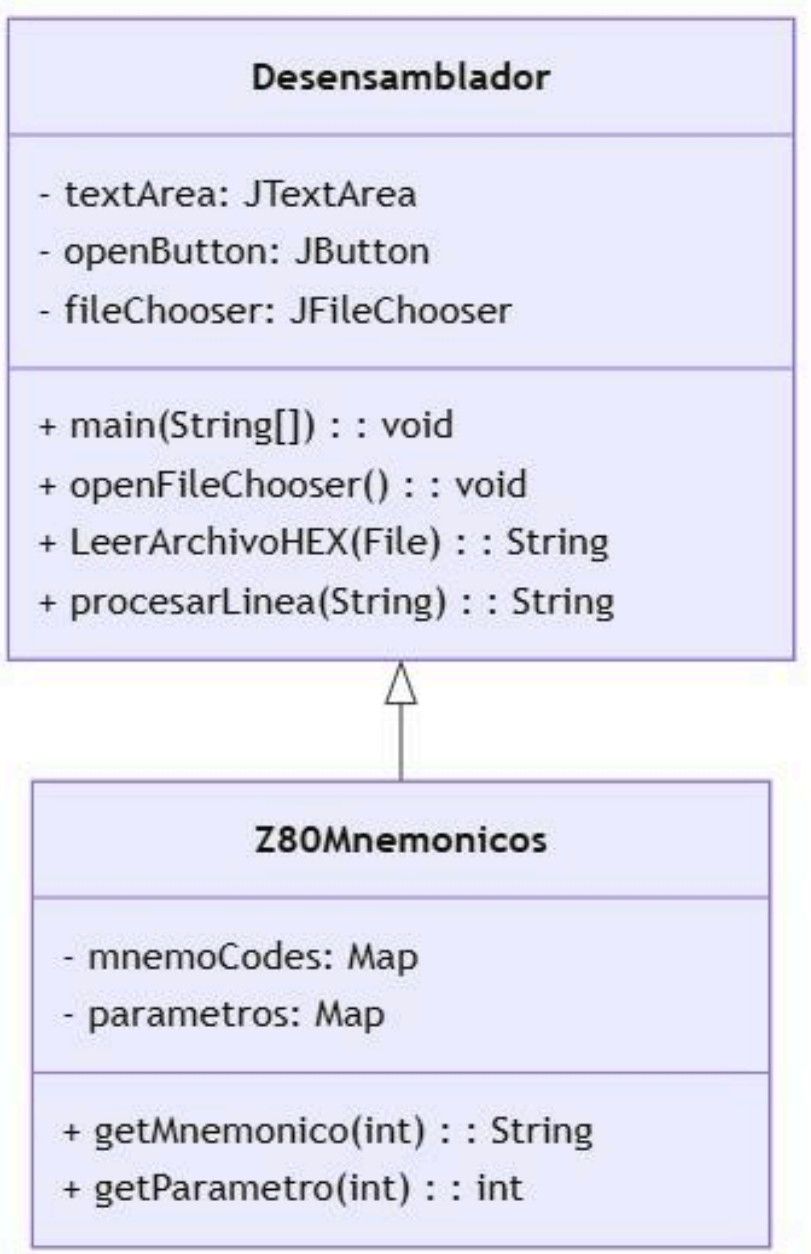
```
    int direccionActual = Integer.parseInt(linea.substring(3, 7), 16);
    int opcode = Integer.parseInt(linea.substring(9, 11), 16);

    String mnemonico = Z80Mnemonicos.getMnemonico(opcode);
    return String.format("%04X: %s", direccionActual, mnemonico);
}
```

Apéndice B: Diagramas

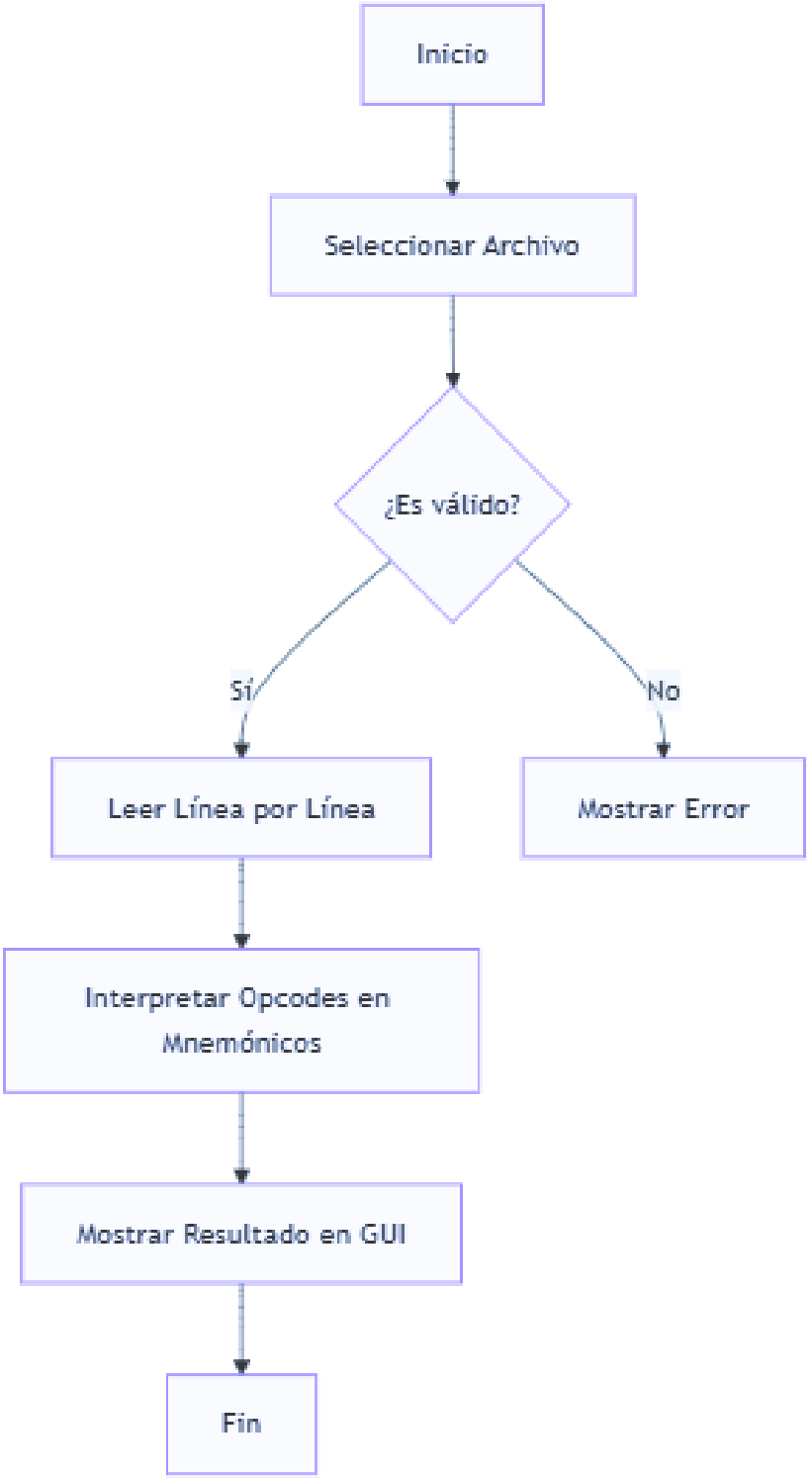
B.1. Diagrama de Clases

Este diagrama de clases muestra cómo se estructuran las clases principales del proyecto y cómo interactúan entre sí:



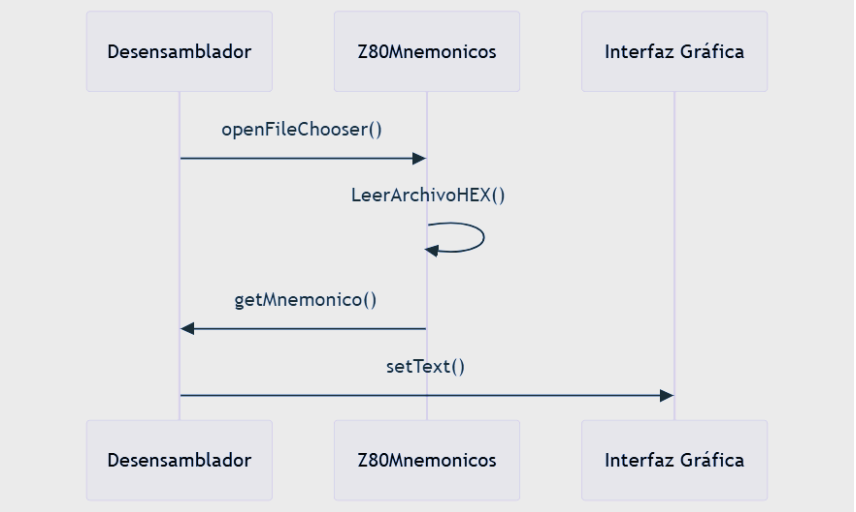
B.2. Diagrama de Flujo

Este diagrama de flujo muestra el proceso desde la selección del archivo hasta la visualización de las instrucciones desensambladas:



B.3. Diagrama de Secuencia

Este diagrama de secuencia muestra cómo las clases interactúan durante el procesamiento de un archivo HEX:



Apéndice C: Resultados de Pruebas

C.1. Plan de Pruebas y Casos de Prueba

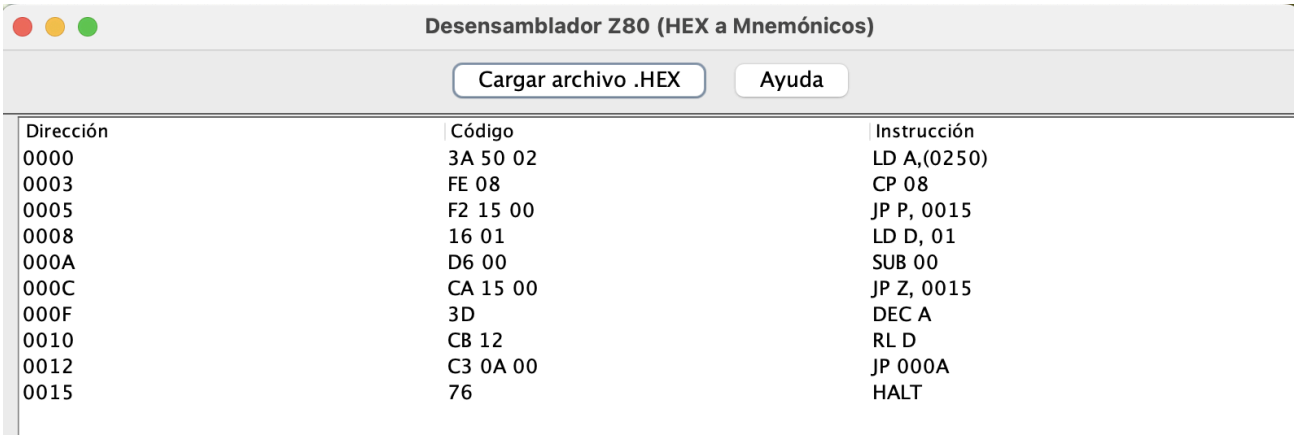
A continuación se detallan los casos de prueba utilizados para verificar el correcto funcionamiento del desensamblador:

ID	Caso de Prueba	Entrada	Resultado Esperado
CP-01	Cargar archivo válido	Archivo HEX válido	Las instrucciones se muestran correctamente en el área de texto.
CP-02	Cargar archivo mal formateado	Archivo con errores en el formato HEX	Mensaje emergente notificando: "Formato de archivo inválido".
CP-03	Procesar archivo con opcodes desconocidos	Archivo HEX con códigos no válidos	Mensaje emergente notificando: "Instrucción no reconocida en la línea X".
CP-04	Procesar archivo vacío	Archivo HEX vacío	Mensaje emergente notificando: "El archivo está vacío".

C.2. Resultados de Ejecución de Pruebas

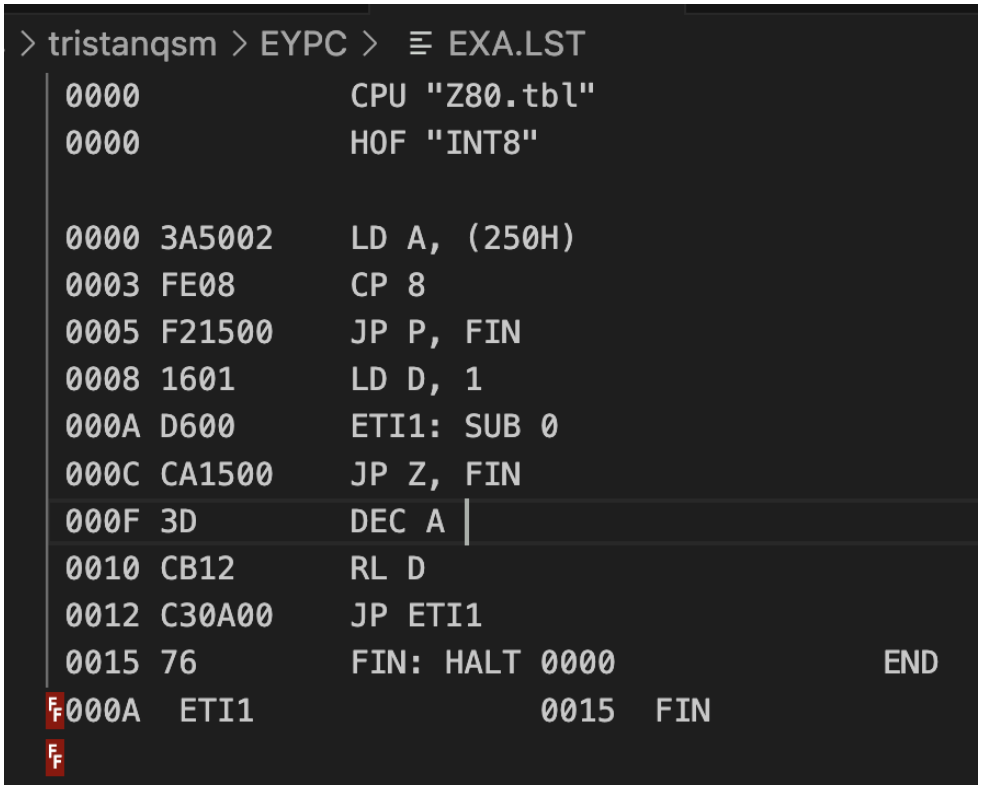
CASO 1:

Usando el Desensamblador Z80



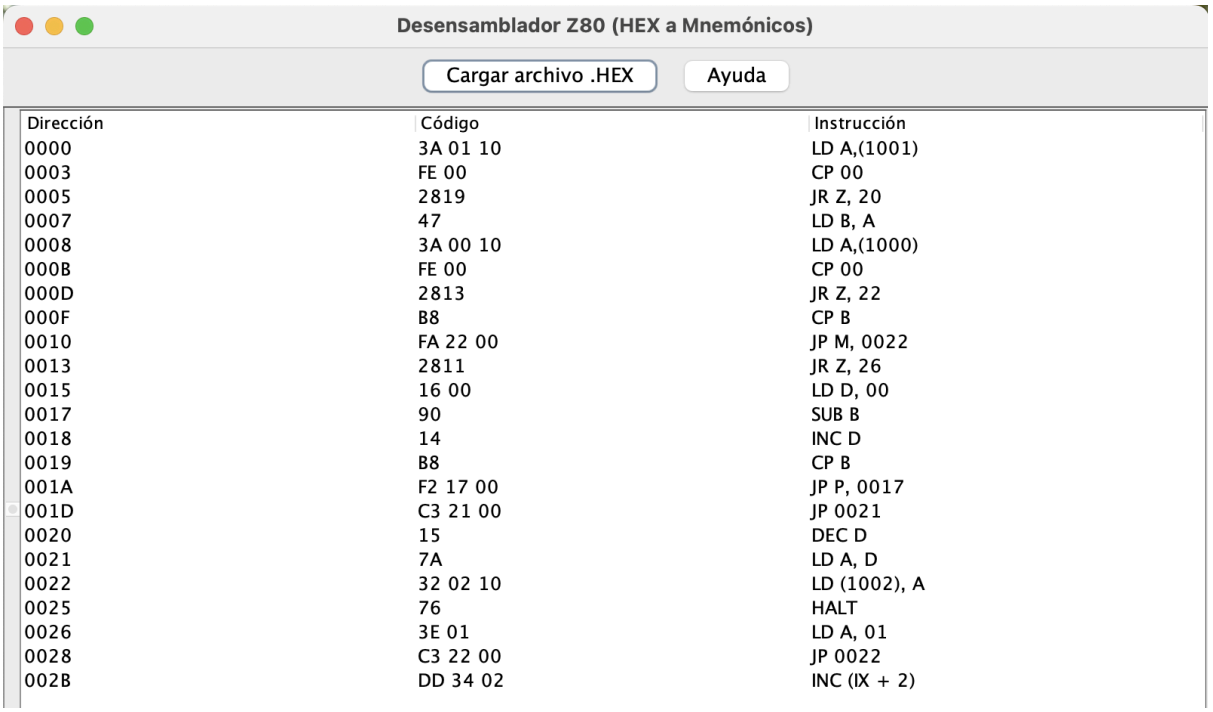
Dirección	Código	Instrucción
0000	3A 50 02	LD A,(0250)
0003	FE 08	CP 08
0005	F2 15 00	JP P, 0015
0008	16 01	LD D, 01
000A	D6 00	SUB 00
000C	CA 15 00	JP Z, 0015
000F	3D	DEC A
0010	CB 12	RL D
0012	C3 0A 00	JP 000A
0015	76	HALT

Comparación con archivo .LST del mismo programa



CASO 2:

Usando el Desensamblador Z80



Desensamblador Z80 (HEX a Mnemónicos)

Cargar archivo .HEX | Ayuda

Dirección	Código	Instrucción
0000	3A 01 10	LD A,(1001)
0003	FE 00	CP 00
0005	2819	JR Z, 20
0007	47	LD B, A
0008	3A 00 10	LD A,(1000)
000B	FE 00	CP 00
000D	2813	JR Z, 22
000F	B8	CP B
0010	FA 22 00	JP M, 0022
0013	2811	JR Z, 26
0015	16 00	LD D, 00
0017	90	SUB B
0018	14	INC D
0019	B8	CP B
001A	F2 17 00	JP P, 0017
001D	C3 21 00	JP 0021
0020	15	DEC D
0021	7A	LD A, D
0022	32 02 10	LD (1002), A
0025	76	HALT
0026	3E 01	LD A, 01
0028	C3 22 00	JP 0022
002B	DD 34 02	INC (IX + 2)

Comparación con archivo .LST del mismo programa

```
tristanqsm > EYPC > ≡ ESTUD1.LST
0000                    CPU "Z80.TBL"
0000                    HOF "INT8"

0000 3A0110                     LD A,(1001H)
0003 FE00                       CP 0
0005 2819                       JR Z, error
0007 47                         LD B,A
0008 3A0010                     LD A,(1000H)
000B FE00                       CP 0
000D 2813                       JR Z, eti1
000F B8                         CP B
0010 FA2200                     JP M, eti1
0013 2811                       JR Z, eti2
0015 1600                       LD D, 0
0017 90                 eti3: SUB B
0018 14                         INC D
0019 B8                         CP B
001A F21700                     JP P,eti3
001D C32100                     JP eti4
0020 15                 error: DEC D
0021 7A                 eti4: LD A,D
0022 320210             eti1: LD (1002H), A
0025 76                         HALT
0026 3E01               eti2: LD A,1
0028 C32200                     JP eti1
002B DD3402                     INC (IX + 2)
```

CASO 3:

Usando el Desensamblador Z80

Desensamblador Z80 (HEX a Mnemónicos)

Cargar archivo .HEX | Ayuda

Dirección	Código	Instrucción
0000	3A 00 10	LD A,(1000)
0003	FE 02	CP 02
0005	FA 32 00	JP M, 0032
0008	4F	LD C, A
0009	0D	DEC C
000A	51	LD D, C
000B	06 00	LD B, 00
000D	DD 21 01 10	LD IX, 1001
0011	1E 00	LD E, 00
0013	DD 7E 00	LD A, (IX + 0)
0016	DD BE 01	CP (IX + 1)
0019	F2 33 00	JP P, 0033
001C	04	INC B
001D	DD 23	INC IX
001F	15	DEC D
0020	7A	LD A, D
0021	FE 00	CP 00
0023	C2 13 00	JP NZ, 0013
0026	7B	LD A, E
0027	FE 00	CP 00
0029	CA 32 00	JP Z, 0032
002C	79	LD A, C
002D	FE 01	CP 01
002F	C2 09 00	JP NZ, 0009
0032	76	HALT
0033	DD 66 01	LD H, (IX + 1)
0036	DD 74 00	LD (IX + 0), H
0039	DD 77 01	LD (IX + 1), A
003C	1E 01	LD E, 01
003E	18DC	JR 1C

Comparación con archivo .LST del mismo programa

tristanqsm > EYPC > BURBUJA.LST

```
0000             CPU "z80.tbl"
0000             HOF "INT8"

0000 3A0010      LD A,(1000H)
0003 FE02        CP 2
0005 FA3200      JP M,eti4
0008 4F          LD C,A
0009 0D          eti5: DEC C
000A 51          LD D,C
000B 0600        LD B,0
000D DD210110    LD IX,1001H
0011 1E00        LD E,0
0013 DD7E00      eti3: LD A, (IX+0)
0016 DDBE01      CP (IX+1)
0019 F23300      JP P,eti2
001C 04          eti1: INC B
001D DD23        INC IX
001F 15          DEC D
0020 7A          LD A,D
0021 FE00        CP 0
0023 C21300      JP NZ,eti3
0026 7B          LD A,E
0027 FE00        CP 0
0029 CA3200      JP Z,eti4
002C 79          LD A,C
002D FE01        CP 1
002F C20900      JP NZ,eti5
0032 76          eti4: HALT
0033 DD6601      eti2: LD H, (IX+1)
0036 DD7400      LD (IX+0),H
0039 DD7701      LD (IX+1),A
003C 1E01        LD E,1
003E 18DC        JR eti1
```

CASO 4:

Usando el Desensamblador Z80

Dirección	Código	Instrucción
0000	3A 00 10	LD A,(1000)
0003	FE 00	CP 00
0005	CA 1E 00	JP Z, 001E
0008	47	LD B, A
0009	3A 00 10	LD A,(1000)
000C	FE 00	CP 00
000E	CA 1E 00	JP Z, 001E
0011	B8	CP B
0012	F2 22 00	JP P, 0022
0015	50	LD D, B
0016	4F	LD C, A
0017	3E 00	LD A, 00
0019	82	ADD A, D
001A	0D	DEC C
001B	C2 1A 00	JP NZ, 001A
001E	32 10 76	LD (7610), A
0021	57	LD D, A
0022	48	LD C, B
0023	C3 19 00	JP 0019

Comparación con archivo .LST del mismo programa

```
0000                    CPU "Z80.tbl"
0000                    HOF "INT8"
0000 3A0010             LD A,(1000H)
0003 FE00               CP 0
0005 CA1E00             JP Z, eti1
0008 47                 LD B, A
0009 3A0010             LD A, (1000H)
000C FE00               CP 0
000E CA1E00             JP Z, eti1
0011 B8                 CP B
0012 F22200             JP P, eti2
0015 50                 LD D, B
0016 4F                 LD C, A
0017 3E00               LD A, 0
0019 82           eti4: ADD A, D
001A 0D           eti3: DEC C
001B C21A00             JP NZ, eti3
001E 320210       eti1: LD (1002H), A
0021 76                 HALT
0022 57           eti2: LD D, A
0023 48                 LD C, B
0024 C31900             JP eti4 0000
```

CASO 5:

Usando el Desensamblador Z80

Desensamblador Z80 (HEX a Mnemónicos)

Cargar archivo .HEX | Ayuda

Dirección	Código	Instrucción
0000	3A 00 20	LD A,(2000)
0003	FD 56 05	LD D, (IY + 5)
0006	DD CB 04 06	RLC (IX + 4)
000A	DD CB 15 0E	RRC (IX + 21)
000E	DD CB 01 1E	RR (IX + 1)
0012	DD CB 06 26	SLA (IX + 6)
0016	DD CB 06 2E	SRA (IX + 6)
001A	DD CB 0C 06	RLC (IX + 12)
001E	DD CB 02 16	RL (IX + 2)
0022	DD CB 17 26	SLA (IX + 23)
0026	DD CB 20 46	BIT 0, (IX + 32)
002A	DD CB 20 4E	BIT 1, (IX + 32)
002E	DD CB 20 56	BIT 2, (IX + 32)
0032	DD CB 20 5E	BIT 3, (IX + 32)
0036	DD CB 20 66	BIT 4, (IX + 32)
003A	DD CB 20 6E	BIT 5, (IX + 32)
003E	DD CB 20 76	BIT 6, (IX + 32)
0042	23	INC HL
0043	DD CB 20 7E	BIT 7, (IX + 32)
0047	DD CB 20 FE	SET 7, (IX + 32)
004B	DD CB 01 DE	SET 3, (IX + 1)

Comparación con archivo .LST del mismo programa

```
0000    3A 00 20        LD   A,(2000H)
0003    FD 56 05        LD   D,(IY+5)
0006    DD CB 04 06     RLC   (IX + 4)
000A    DD CB 15 0E     RRC   (IX+21)
000E    DD CB 01 1E     RR   (IX+1)
0012    DD CB 06 26     SLA   (IX+6)
0016    DD CB 06 2E     SRA   (IX+6)
001A    DD CB 0C 06     RLC   (IX+12)
001E    DD CB 02 16     RL   (IX+2)
0022    DD CB 17 26     SLA   (IX+23)
0026    DD CB 20 46     BIT   0,(IX+32)
002A    DD CB 20 4E     BIT   1,(IX+32)
002E    DD CB 20 56     BIT   2,(IX+32)
0032    DD CB 20 5E     BIT   3,(IX+32)
0036    DD CB 20 66     BIT   4,(IX+32)
003A    DD CB 20 6E     BIT   5,(IX+32)
003E    DD CB 20 76     BIT   6,(IX+32)
0042    23              INC   HL
0043    DD CB 20 7E     BIT   7,(IX+32)
0047    DD CB 20 FE     SET   7,(IX+32)
004B    DD CB 01 DE     SET   3,(IX+1)
```

Total de Casos de Prueba: 5

Pruebas Exitosas: 5

Observaciones Generales: El sistema funcionó correctamente en la mayoría de los casos, aunque el manejo de opcodes desconocidos necesita una mejora para ofrecer mensajes de error más claros.

Apéndice D: Detalles del Algoritmo de Mnemónicos

El algoritmo para convertir opcodes en mnemónicos fue implementado mediante una serie de mapas (HashMap), donde las claves son los códigos de operación y los valores son los mnemónicos asociados. Este sistema permite una traducción rápida y eficiente, compatible con las instrucciones estándar y extendidas del Z80.

Anexo A: Consideraciones de Mantenimiento

Para garantizar la sostenibilidad del sistema a largo plazo, se estableció un plan de mantenimiento que incluye correcciones de errores, adaptaciones a nuevas versiones de Java, y la posibilidad de agregar nuevas instrucciones Z80 según sea necesario. Además, se incorporaron prácticas de mantenimiento preventivo, como la actualización periódica de bibliotecas y la realización de auditorías de código.