Laboratorio 1 Arquitectura Von Neuman

Equipo de Trabajo

Carlos Osvaldo Zapata Arango 1044504411

Alejandro López Ospina 1058847429

Arquitectura de Computadores

Asesor

Yohany Ortiz

Departamento de Sistemas

Facultad de ingeniería

Universidad de Antioquia

2024-1

Índice

Objetivo

El Objetivo de este trabajo es aplicar los conceptos indicados en video de referencia: “*Suma sencilla Simulador máquina Von Neumann*” e implementar dicho simulador en un lenguaje de programación mostrando su funcionamiento." Este video, publicado por Abadía digital el 22 de agosto de 2022, ofrece una explicación detallada de cómo se realiza una suma sencilla utilizando este simulador."

Herramientas utilizadas

Entorno de Desarrollo Integrado (IDE):

- NetBeans 8.2: Se utilizó NetBeans como entorno de desarrollo integrado (IDE) principal para escribir, compilar y depurar el código del programa. NetBeans proporciona una amplia gama de características y herramientas para el desarrollo de software en Java, incluyendo la edición de código, la gestión de proyectos, la depuración y la integración con sistemas de control de versiones.

Lenguaje de Programación:

- Java: Java se utilizó como el lenguaje de programación principal para desarrollar el programa de simulación. Otros Recursos:
- Librerías de Java: Se aprovecharon las librerías estándar de Java para implementar funcionalidades específicas del programa, como la manipulación de cadenas, la gestión de entrada/salida y la manipulación de estructuras de datos.

Descripción

Funcionamiento

Tabla de información de instrucciones

instrucción	Decodificador	Operación	Que hace la instruccion
0000	+	Suma	Acu+Mem→acu
0001	-	Resta	Acu-Memà→acu
0010	*	Producto	Acu*Mem→àacu
0011	´	Exponente	Acu´Mem→acu
0100	&	Operador AND	Acu AND Mem→acu
0101	|	Operador OR	Acu OR Mem-->acu
0110	M	Mover a memoria	Acu à Mem
0111	----	Finalizar	Acaba el programa

En el ejercicio actual se implementó un simulador de máquina de Von Neumann, el cual toma como referencia la suma de 2 números, describiendo paso a paso a nível de procesos dentro la máquina el proceso realizado, desde la asignación de la operación, para este caso en específico sería la suma, hasta el almacenamiento del resultado en memoria; lo anterior, realizando como primer paso la asignación de un código binario relacionado diferentes operaciones, como el proceso de suma, de guardado en memoria y el código asociado a la finalización del proceso.

El programa toma 2 números preestablecidos los cuales se almacenan en los campos de memoria posteriores a la asignación de las operaciones antes mencionadas, todo lo anterior agrupado en una tabla de memoria que relaciona una dirección de memoria a una operación o número establecido, iniciando por la identificación de la suma, el cual dirige la instrucción a la dirección de memoria donde se encuentra el primer número en formato binario y posterior a ello registra el dato y luego lo suma en una variable “acumuladora”, identificando otra operación de suma, repitiendo el proceso con el segundo número, hallando en la variable acumuladora el total de la suma de ambos números.

Finalmente, posterior a realizar la operación y almacenarla provisionalmente en la variable acumuladora, pasa a la instrucción de memoria, en la cual el programa asigna a una dirección dentro de la tabla de memoria el resultado de la operación realizada para terminar con la instrucción dentro de la tabla asociada a la finalización del proceso.

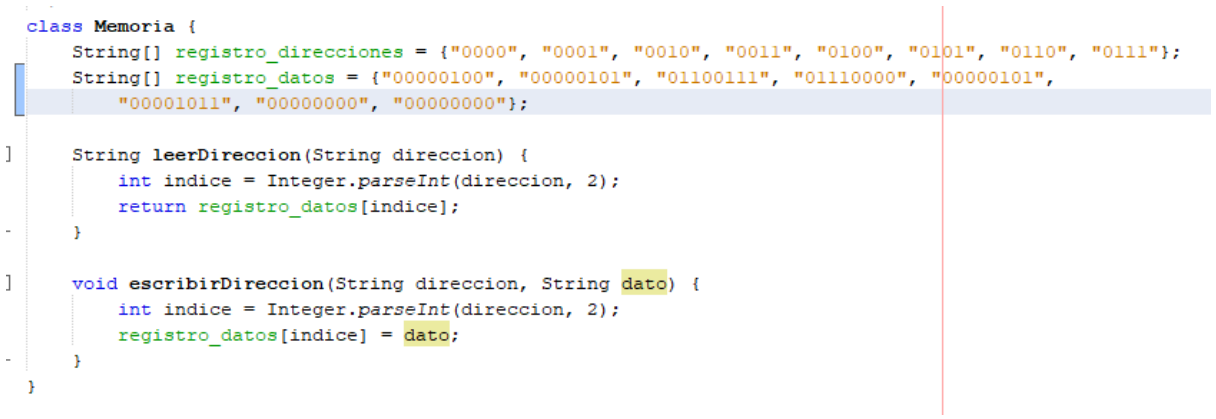
Descripción de las clases:

Clase Memoria:

Esta clase representa la memoria de la máquina de Von Neumann. Contiene dos arreglos de cadenas de texto: uno para las direcciones de memoria y otro para los datos almacenados en esas direcciones. Proporciona métodos para leer y escribir datos en la memoria.

Tabla de memoria:

Direccion	Contenido	4 primeros bits
0001	00000100	Suma
0010	00000101	Suma
0010	01100111	Mover a la memoria
0011	01110000	Finalizar
0100	00000111	Tenemos el 5
0101	00001011	Tenemos el 11
0110	00000000	No los utilizamos
0111	00000000	No los utilizamos



```
class Memoria {
    String[] registro_direcciones = {"0000", "0001", "0010", "0011", "0100", "0101", "0110", "0111"};
    String[] registro_datos = {"00000100", "00000101", "01100111", "01110000", "00000101",
        "00001011", "00000000", "00000000"};

    String leerDireccion(String direccion) {
        int indice = Integer.parseInt(direccion, 2);
        return registro_datos[indice];
    }

    void escribirDireccion(String direccion, String dato) {
        int indice = Integer.parseInt(direccion, 2);
        registro_datos[indice] = dato;
    }
}
```

- Atributos:
    - **Registro_direcciones:** Un arreglo de cadenas de texto que representa las direcciones de memoria.
    - **Registro_datos:** Un arreglo de cadenas de texto que representa los datos almacenados en las direcciones de memoria correspondientes.
- Métodos:
    - **LeerDireccion(String direccion)**: Retorna el dato almacenado en la dirección de memoria especificada.

- **EscribirDireccion(String direccion, String dato):** Escribe el dato especificado en la dirección de memoria indicada.

Clase UnidadAritmeticoLogica:

Esta clase representa la unidad aritmético lógica de la máquina de Von Neumann. Contiene un acumulador y un registro de entrada, ambos representados como cadenas de texto.

```
class UnidadAritmeticoLogica {
    String acumulador = "00000000";
    String registro_entrada = "00000000";
}
```

- Atributos:
  - **Acumulador:** Una cadena de texto que representa el acumulador de la unidad aritmético lógica.
  - **Registro_entrada**: Una cadena de texto que representa el registro de entrada de la unidad aritmético lógica.

Clase UnidadControl:

Esta clase representa la unidad de control de la máquina de Von Neumann. Contiene una referencia a la memoria (decodificador), así como el contador de programa y el registro de instrucciones, ambos representados como cadenas de texto.

```
class UnidadControl {
    Memoria decodificador;
    String contador_programa = "0000";
    String registro_instrucciones = "00000000";

    UnidadControl(Memoria decodificador) {
        this.decodificador = decodificador;
    }
}
```

- **Atributos:**
    - **Decodificador:** Una referencia a la memoria.
    - **Contador_programa:** Una cadena de texto que representa el contador de programa de la unidad de control.
    - **Registro_instrucciones:** Una cadena de texto que representa el registro de instrucciones de la unidad de control.

Clase Main (Clase Principal):

Esta clase contiene el método principal (main) y se utiliza para iniciar la aplicación de la interfaz gráfica. Se encarga de crear instancias de las clases Memoria, UnidadAritmeticoLogica y UnidadControl, así como de manejar la interfaz gráfica para ejecutar el programa y mostrar los resultados.

```
public class Main extends Application {
    Memoria memoria;
    UnidadControl unidadControl;
    UnidadAritmeticoLogica unidadAritmeticoLogica;
    TextArea seguimientoTextArea;
    TextArea resultadoTextArea;
    StringBuilder numerosSumados;

 @Override
public void start(Stage primaryStage) {...30 lines }


    private void ejecutarPrograma() {...57 lines }

private void mostrarResultado() {...5 lines }



    public static void main(String[] args) {
        launch(args);
    }
}
```

- Métodos:
    - **Start(Stage primaryStage):** Configura la interfaz gráfica y define los elementos necesarios para la ejecución del programa.
    - **EjecutarPrograma():** Ejecuta el programa de la máquina de Von Neumann y muestra el seguimiento de la ejecución en un TextArea.
    - **MostrarResultado():** Muestra el resultado de la ejecución del programa en otro TextArea.

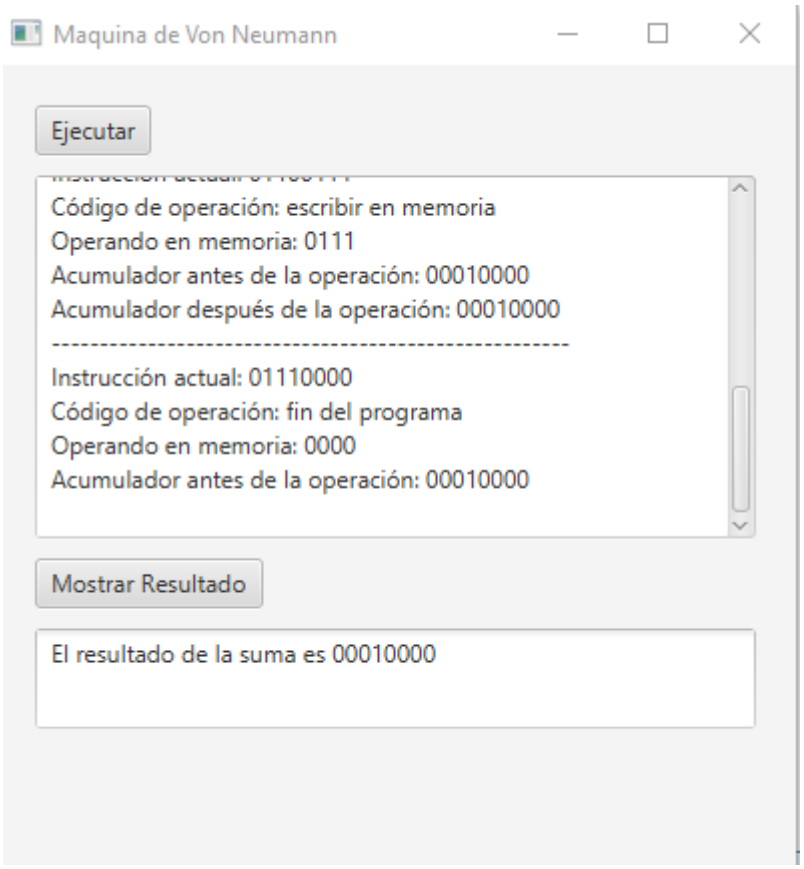
Resultados

Como resultado de la implementación del programa, se logra obtener de forma de “paso a paso” todo el proceso descrito en la descripción del laboratorio, en el cual se puede evidenciar de forma descriptiva la transición dentro de la tabla de memoria y el resultado de realizar la suma y almacenar este resultado en un campo de memoria.

Una vez ejecutada la aplicación nos mostrará el siguiente pantallazo.

Una vez iniciado el proceso por medio del botón “ejecutar”, se puede identificar la secuencia de ejecución donde se evidencia el paso a paso desde que establece la operación de suma, hasta que realiza la operación y la almacena en el campo de

memoria asignado, para posterior al presionar el botón “Mostrar Resultado”, arroje el valor resultante de la operación realizada.



Análisis de resultados

Con la ejecución del programa se puede evidenciar que posterior a tener unos números preestablecidos dentro de la tabla de memoria, se ejecuta de manera correcta, realizando el proceso de manera exitosa y arrojando el resultado de la suma.

Sin embargo nos enfrentamos ante una dificultad la cual en el momento no logramos solucionar de buena manera, y es en los casos en los cuales queríamos que el programa pudiese solicitar al usuario los números iniciales a operar y realizar el proceso con números aleatorios, logrando así dinamizar el programa y haciéndolo

más eficiente, resaltando que su función u objetivo principal se logra obtener de manera correcta.

Con la implementación de este algoritmo, nos permitió identificar el proceso que debe realizar las máquinas ante una operación que consideramos básica, donde se puede evidenciar como inicialmente por medio de lenguaje binario debe asignar las acciones u operaciones a realizar, para posteriormente cumplir su objetivo que para este caso práctico es la suma de 2 números enteros.

Referencias

- Oracle Corporation. "Getting Started with NetBeans IDE". Documentación oficial de NetBeans 8.2. Disponible en: https://docs.oracle.com/netbeans/nb82/netbeans/NBDAG/gs_nbeans.htm#NBDAG111.
- Abadía digital. (22 de agosto de 2022). Suma sencilla Simulador máquina Von Neumann [Archivo de video]. YouTube. Disponible en: https://www.youtube.com/watch?v=WYuDSP_Rffg&t=82s.
- Oracle. (s.f.). Java SE Technologies. Recuperado el 21 de febrero de 2024, de https://docs.oracle.com/javase/8/javase-clienttechnologies.htm
- Repositorio del Proyecto https://github.com/zapata-git/MaquinaDeVonNeuman.git