LABORATORIO 2 - ORDENAMIENTO COMBSORT EN MIPS

ANDRÉS FELIPE GIRALDO YUSTI[1] **y JUAN CAMILO VELEZ**[2]

[1] *Universidad de Antioquia, Facultad de Ingeniería , andres.giraldoy@udea.edu.co*
[2] *Universidad de Antioquia, Facultad de Ingeniería , camilo.velez4@udea.edu.co*

JOHN BYRON BUITRAGO PANIAGUA

1. Introducción

En esta práctica se desarrolló un programa en lenguaje ensamblador MIPS que permite leer un archivo de texto con números separados por comas, luego los ordena utilizando el algoritmo CombSort en orden ascendente, y después guardar el resultado ordenado en un nuevo archivo de texto llamado lista-ordenada.txt.

El objetivo principal es aplicar conceptos de bajo nivel como el acceso a memoria y manipulación de registros, evitando utilizar pseudo-instrucciones para trabajar directamente con el conjunto de instrucciones básicas de MIPS.

2. Planteamiento del problema y objetivos

2.1. Problema

Desarrollar un programa en MIPS que:

- Lea un archivo con números enteros separados por coma.
- Procese los datos y los almacene en memoria.
- Ordene el arreglo utilizando el algoritmo CombSort de forma ascendente.
- Escriba el resultado en un nuevo archivo de salida, separados por coma.

2.2. Objetivos

- Implementar entrada/salida de archivos mediante llamadas al sistema.
- Implementar desde cero el algoritmo CombSort en MIPS sin pseudo-instrucciones.
- Escribir el resultado final en un nuevo archivo.

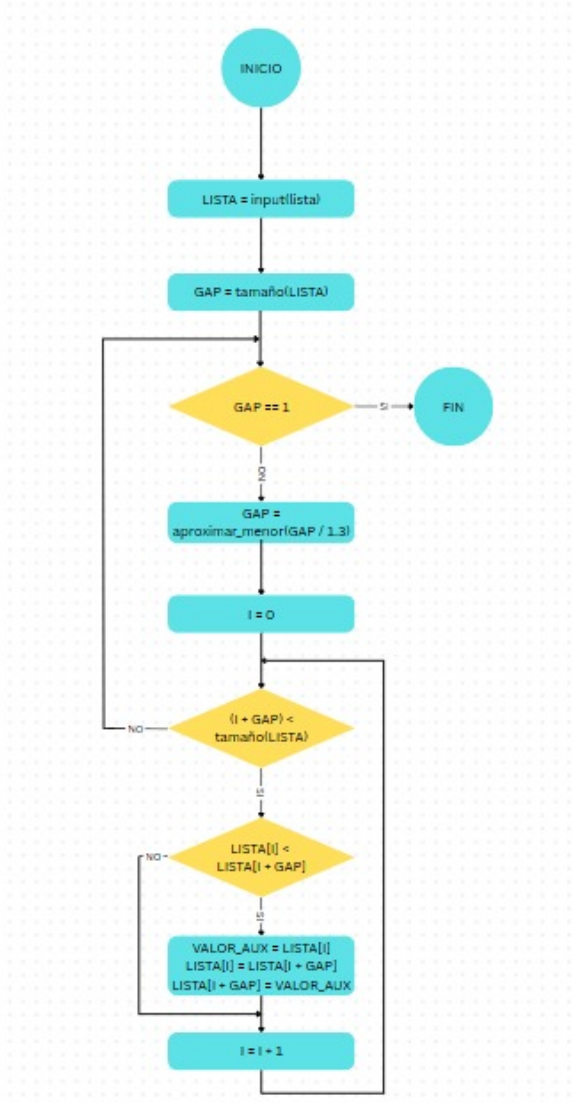

Figura 1. Diagrama de flujo del algoritmo de ordenamiento CombSort

3. Fundamentos teóricos

CombSort es un algoritmo de ordenamiento basado en la idea de comparar elementos de un arreglo que están separados por una distancia determinada, llamada "gap". En cada iteración, se comparan los elementos ubicados a esa distancia y se intercambian si están desordenados. Luego, el "gap"se reduce progresivamente hasta llegar a uno, momento en el que se realizan comparaciones entre elementos contiguos.

El algoritmo permite recorrer y ordenar los datos de forma eficiente, siguiendo una lógica estructurada de comparaciones repetidas mientras se reduce el "gap". Esta mecánica se puede implementar mediante ciclos anidados y condiciones de intercambio.

En esta práctica, se implementó CombSort en lenguaje ensamblador MIPS, utilizando el entorno MARS. Se manejaron operaciones como la lectura y escritura de archivos mediante llamadas al sistema (syscalls), y se utilizaron registros para almacenar temporalmente datos.

4. Metodología

El desarrollo de esta práctica se realizó construyendo y probando cada parte del programa en MIPS paso a paso, con el objetivo de leer un archivo, procesar sus datos numéricos, ordenarlos y escribir el resultado en otro archivo.

La Figura 2 muestra el flujo lógico general que se implementó para lograr el objetivo del laboratorio. Las fases principales de la metodología aplicada fueron:

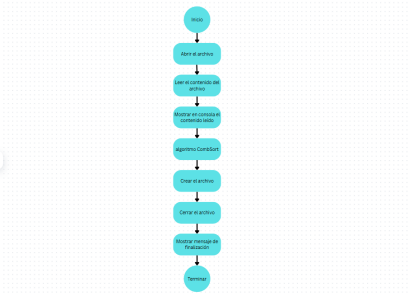

Figura 2. Diagrama de flujo del algoritmo

- **Lectura del archivo**: Se abrió el archivo que contiene los números y se leyó su contenido.
- **Parseo de los datos**: se recorrió el contenido leído, identificando los números según el separador definido y convirtiéndolos a enteros que fueron almacenados en un arreglo en memoria.
- **Ordenamiento**: se aplicó el algoritmo CombSort directamente sobre el arreglo cargado, comparando y reordenando los elementos mediante ciclos y saltos condicionales en MIPS.
- **Escritura del archivo de salida**: tras ordenar los datos, se generó un nuevo archivo, escribiendo allí los números ordenados como texto decimal.

Cada etapa fue probada individualmente para asegurar su correcto funcionamiento antes de integrarse al flujo completo. Se puso especial atención en el manejo de memoria alineada y en el uso correcto de los syscalls para interactuar con archivos en MIPS bajo el entorno MARS.

5. Diseño del sistema

El sistema desarrollado en MIPS Assembly permite leer una lista de números desde un archivo de texto, almacenarlos en un arreglo en memoria, aplicar el algoritmo de ordenamiento **CombSort**, y finalmente guardar el resultado en un nuevo archivo. A continuación, se describen los bloques funcionales y su interacción general.

5.1. Lectura del archivo

El programa comienza solicitando la apertura del archivo `lista.txt`, ubicado en el mismo directorio del ejecutable. Para esto se utiliza la syscall correspondiente a `open` en modo lectura. El contenido se carga en un *buffer* usando la syscall `read`, el cual posteriormente es mostrado en consola para verificación.

5.2. Parseo de datos

Una vez leído el contenido, se realiza un recorrido carácter por carácter del *buffer*. Se construyen números decimales detectando separadores (comas) y se almacenan en una estructura de datos en memoria utilizando direcciones alineadas. Este proceso convierte cada secuencia de caracteres ASCII en su equivalente numérico, utilizando multiplicaciones sucesivas por 10 y sumas de dígitos.

5.3. Ordenamiento CombSort

Sobre el arreglo de enteros se aplica el algoritmo CombSort. Este algoritmo mantiene una variable gap inicializada como el número de elementos (n) y en cada iteración la reduce según la fórmula:

$$gap = \left\lfloor \frac{gap}{1{,}3} \right\rfloor$$

El algoritmo compara elementos con un espacio de gap entre ellos, intercambiándolos si es necesario. Se repite el proceso hasta que $gap = 1$ y no se hayan hecho más intercambios, garantizando que el arreglo está ordenado.

5.4. Creación del archivo de salida

Una vez ordenado el arreglo, se abre un nuevo archivo con nombre `lista_ordenada.txt` utilizando la syscall `open` en modo escritura con permisos de creación. Cada número del arreglo se convierte nuevamente a ASCII, carácter por carácter, y se escribe en el archivo seguido de una coma.

5.5. Visualización y validación

El sistema imprime el contenido original del archivo y el número total de elementos parseados. También puede imprimirse el arreglo ordenado por consola si se habilita dicha sección en el código. Esto permite validar la correcta ejecución del proceso completo sin necesidad de abrir los archivos manualmente.

6. Resultados y análisis

El sistema logró leer correctamente archivos de entrada como ‘lista.txt‘ con contenido tipo: ‘12,8,15,4,21‘, ordenarlos y generar el archivo ‘lista_ordenada.txt‘ con los valores: ‘4,8,12,15,21‘.

Se detectaron y corrigieron errores comunes mal uso de syscall y conversión incorrecta de caracteres. El sistema fue validado progresivamente por partes para garantizar estabilidad y comprensión del código.

7. Conclusiones

Este laboratorio permitió poner en práctica conceptos esenciales de programación en ensamblador, manejo de memoria, archivos y estructuras algorítmicas. La implementación del algoritmo CombSort sin pseudo-instrucciones representó un reto significativo que consolidó el entendimiento de bajo nivel en la arquitectura MIPS.

Además, se reforzó la importancia de las conversiones entre tipos de datos y el uso correcto de las llamadas al sistema, aspectos clave en la programación de sistemas embebidos o de bajo nivel.