Ordenamiento y matemáticas discretas

Camila Gárate (6 horas) , Benjamin Sandoval (6 horas)
Departamento de Ingeniería Informática
Universidad de Santiago de Chile, Santiago, Chile
camila.garate@usach.cl y benjamin.sandoval.b@usach.cl

***Resumen*—Este informe presenta el desarrollo de dos algoritmos en lenguaje ensamblador MIPS, cada uno destinado a resolver diferentes problemas. El primer algoritmo aborda el desafío de descifrar un mensaje alienígena recibido por un estudiante mediante la implementación de un algoritmo de ordenamiento. Utilizando el simulador MARS, se solicita al usuario datos numéricos que son almacenados y ordenados en el segmento de datos de MIPS. El segundo algoritmo, diseñado para mejorar la comprensión de la matemática discreta, incluye cálculos de combinatoria, permutación y variación. Un menú interactivo en ensamblador MIPS permite a los usuarios seleccionar y realizar estos cálculos, utilizando técnicas alternativas a las instrucciones de multiplicación y división. Los resultados mostraron que ambos algoritmos funcionaron correctamente, demostrando su efectividad en sus respectivos contextos. La principal contribución de este trabajo es la demostración práctica de conceptos de arquitectura de computadores y programación en ensamblador, destacando la importancia del ordenamiento de datos y los cálculos matemáticos.**

***Palabras claves*—Selection Sort, MIPS, Matemática discreta, Subrutinas.**

I. Introducción

Salfate, un estudiante de la Universidad de Santiago de Chile, tuvo un encuentro con seres alienígenas que le transmitieron un mensaje codificado. Incapaz de comprenderlo, recurre a los estudiantes de ingeniería informática para descifrarlo. Para resolver este problema, se desarrollará un algoritmo de ordenamiento en lenguaje ensamblador MIPS. Utilizando el simulador MARS, se solicitará al usuario ingresar hasta 8 números enteros, que seran almacenados y ordenados en el segmento de datos de MIPS mediante subrutinas modulares.

Paralelamente, Thomas Riffo, un ayudante de estadística computacional, se enfrenta a su primera sesión de tutoría de matemática discreta. Para hacer la clase más dinámica, sus amigos Reinaldo y Felipe proponen implementar cálculos de combinatoria, permutación y variación en ensamblador MIPS. Los estudiantes diseñarán un menú interactivo que permitirá seleccionar la operación matemática deseada, mejorando así la comprensión de la materia. Los objetivos de estas actividades serán aplicar conceptos de arquitectura de computadores, fortalecer las habilidades en programación de bajo nivel y proporcionar herramientas prácticas para el aprendizaje de matemática discreta.

II. Antecedentes

En este laboratorio se asume que las definiciones de algunos conceptos que ya se conocen por el laboratorio anterior [1], los conceptos necesarios para realizar este laboratorio son los siguientes:

II-A. Algoritmo de Selección

El algoritmo de selección es un método de ordenamiento que funciona seleccionando repetidamente el elemento más grande (o más pequeño) de la lista y colocándolo en su posición correcta. El proceso se repite para el resto de la lista hasta que todos los elementos esten ordenados. Este algoritmo tiene una complejidad de tiempo de $O(n^2)$, lo que lo hace adecuado para listas pequeñas. [2]

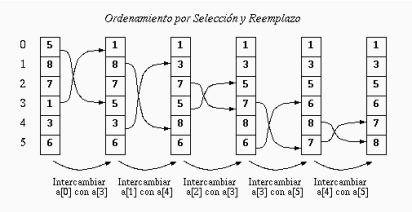

Figura 1: (Representación básica del algoritmo de selección. El algoritmo usado tiene variaciones, como por ejemplo, que se realiza buscando el mayor elemento y que existe un paso intermedio donde se reemplaza ese elemento por cero utilizando una lista auxiliar [3])

II-B. Matemática Discreta

La matemática discreta incluye el estudio de estructuras matemáticas que son fundamentalmente discretas en lugar de continuas. En el contexto de este trabajo, se abordaron tres conceptos clave:

II-B1. Combinatoria: Se refiere al conteo de combinaciones posibles de un conjunto de elementos. [4]

$$C_{n,m} = \binom{n}{m} = \frac{n!}{m!\cdot(n-m)!}$$

Figura 2: (Fórmula Combinatoria)

II-B2. Permutación: Implica el reordenamiento de todos los elementos de un conjunto en todas las formas posibles.[4]

II-B3. Variación: Se refiere a las diferentes maneras en que se pueden seleccionar y ordenar un subconjunto de elementos de un conjunto más grande. [4]

$$Pn = n!$$

Figura 3: (Fórmula Permutación)

$$V_m^n = \frac{m!}{(m-n)!}$$

Figura 4: (Fórmula Variación)

II-C. Subrutinas

El uso de subrutinas en programación en ensamblador permite una estructura modular y organizada del código. Las subrutinas facilitan la reutilización del codigo y el mantenimiento del programa al separar las funciones en bloques manejables. [5]

```
main:
 # Muestra el menú en consola
 la $a0, menu
 jal imprimir_mensaje

 # Leer la opción del usuario
 jal solicitar_entero
 move $t0, $v0

 # Verificar la opción seleccionada
 beq $t0, 1, opcion1
 beq $t0, 2, opcion2
 beq $t0, 3, opcion3
 j opcion_no_valida
```

Figura 5: (Subrutina main. Su uso corresponde a una buena práctica de programación)

III. Materiales y Métodos

III-A. Materiales

1. Sistema operativo: Windows 8.1 y 10
2. Procesador: Intel Pentium (4 núcleos) y Intel Core i5-4310u
3. MARS v4.5 (ensamblador de MIPS)
4. Software LaTeX, editor Overleaf

III-B. Métodos

Algoritmo de Ordenamiento:

El programa comienza solicitando al usuario que ingrese los elementos de la lista, estos elementos se almacenan en la memoria comenzando desde la dirección '0x100100a0'. Luego, se hace una copia de los datos en la dirección 0x100100c0 donde se modifica la lista de acuerdo al algoritmo.

Se inicializa un contador 'i' en 0, se establece un bucle principal ('whileMain') que itera mientras 'i' sea menor o igual a 7, lo que permite ordenar hasta 8 elementos. En cada iteración del bucle se llama a la subrutina 'defineMayor', que compara todos los elementos de la lista para encontrar el mayor elemento y lo carga en el registro'$s0', después se

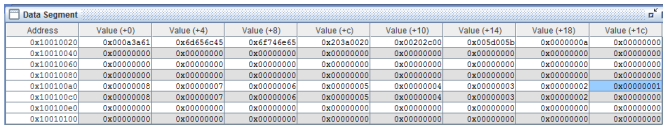

Figura 6: (Copia de la lista ejemplo [8,7,6,5,4,3,2,1] en la dirección 0x100100c0 del data segment)

llama a la subrutina 'reemplazarMayorPorCero', que reemplaza el mayor elemento encontrado con 0 en la lista original y guarda el mayor en una nueva lista ubicada en la dirección '0x100100e0', luego se incrementa el contador 'i' en 1. El bucle se repite hasta que todos los elementos de la lista estén ordenados.

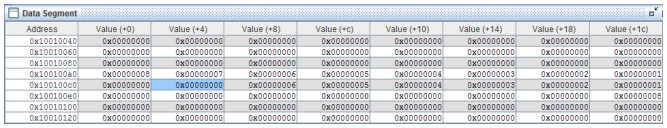

Figura 7: (En este punto, se han encontrado los 2 valores mayores de la lista. Se reemplazan por ceros en 0x100100c0 y se ubican en las últimas posiciones de 0x100100e0)

Se inicializa un segundo bucle ('whileMain2') para imprimir la lista ordenada. En cada iteración del bucle se llama a la subrutina 'mostrarResultado' para imprimir cada uno de los elementos ordenados, después se llama a la subrutina 'imprimirComa' para imprimir comas separadoras entre los elementos, y luego se incrementa el contador 'i' en 1. El bucle se repite hasta que se impriman todos los elementos, y se imprime el corchete de cierre para indicar el final de la lista.

```
Elemento 6: 3
Elemento 7: 2
Elemento 8: 1
[1, 2, 3, 4, 5, 6, 7, 8]
-- program is finished running --
```

Figura 8: (Visualización de resultado en consola de la lista ejemplo)

Matemática Discreta:

El programa comienza mostrando un menú interactivo que permite seleccionar entre 3 operaciones matemáticas: combinatoria, permutación y variación. Este menú se presenta al usuario mediante una cadena de texto almacenada en la memoria, la cual se imprime en consola. El usuario ingresa una opción numérica correspondiente a la operación deseada. Dependiendo de la opción seleccionada por el usuario, el programa ejecuta una de las siguientes subrutinas:

1. 'opcion1': Cálculo de combinatoria
2. 'opcion2': Cálculo de permutaciones
3. 'opcion3': Cálculo de variaciones

Si la opción ingresada no es válida, se imprime un mensaje de error y el programa vuelve al menú principal.

```
Seleccione la fórmula:
1. Combinatoria
2. Permutaciones
3. Variaciones
Seleccione una opción:
4
No válido
```

Figura 9: (Mensaje de error tras ingresar un 4 (entrada no válida))

Para el cálculo de combinatoria, se solicita al usuario ingresar los valores de ‘n’ y ‘m’, se verifica que ambos sean mayores o iguales a 0 y que ‘n’ sea mayor o igual a ‘m’, después se calculan el numerador y el denominador para la fórmula de combinatoria utilizando subrutinas para realizar operaciones aritméticas básicas.

```
Seleccione una opción:
1
n?=6
m?=3
Resultado: 20
-- program is finished running --
```

Figura 10: (Ejemplo de uso para el cálculo de la combinatoria entre 6 y 3)

Para el cálculo de permutaciones, se solicita al usuario ingresar el valor de ‘n’, se verifica que este sea mayor o igual a 0 y luego se calcula ’n!’ utilizando una subrutina de permutación que hace uso del segmento de datos en la dirección 0x100100a0.

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x656c6553	0x6f696363	0x6c20656e	0xf3662061	0x6c756d72	0x310a3a61	0x6f43202e	0x6e69626d
0x10010020	0x726f7461	0x320a6169	0x6550202e	0x74756d72	0x6f696361	0x0a73656e	0x56202e33	0x61697261
0x10010040	0x6e6f6963	0x330a7365	0x63656e65	0x6e6f6963	0x6e752065	0x706f2061	0x6ef36963	0x6e000a3a
0x10010060	0x6d003d3f	0x4e003d3f	0xe176206f	0x6f64696c	0x206f4e00	0x696ce176	0x0a006f64	0x73655200
0x10010080	0x61746c75	0x203a6f64	0x00000000	0x00000000	0x3f800000	0x3dcccccd	0x3c23d70a	0x3a83126f
0x100100a0	0x00000006	0x00000005	0x00000004	0x00000003	0x00000002	0x00000001	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Figura 11: (Uso del segmento de datos para el cálculo de una permutación lineal con n = 6, dirección 0x100100a0)

Para el cálculo de variaciones, el usuario ingresa los valores de ‘m’ y ‘n’, se verifica que ambos sean mayores o iguales a 0 y que ‘m’ sea mayor o igual a ‘n’, se calculan las variaciones utilizando las subrutinas para el cálculo de factoriales y operaciones aritméticas. Solicitar ’m’ primero que ’n’ se debe a que se intenta imitar el comportamiento de una calculadora CASIO modelo fx-350LA PLUS que recibe las entradas de datos en este orden.

Figura 12: (Calculadora CASIO fx-350LA PLUS)

IV. Resultados

Algoritmo de Ordenamiento:

Los elementos ingresados se almacenaron correctamente en la memoria desde la dirección ’0x100100a0’, la subrutina ’guardarListaEnMemoria’ capturó y almacenó eficientemente los datos ingresados por el usuario.

El algoritmo identificó y posicionó correctamente el mayor elemento en cada iteración, reemplazando con 0 y almacenándolo en ’0x100100e0’. Las subrutinas ’defineMayor’ y ’reemplazarMayorPorCero’ funcionaron eficazmente para ordenar la lista, mejorando la claridad.

La lista ordenada se imprimió correctamente en la consola, las subrutinas ’mostrarResultado’ e ’imprimirComa’ formatearon la salida, facilitando la verificación del resultado.

Matemática Discreta:

El menú se presentó correctamente, permitiendo seleccionar entre combinatoria, permutación y variación. La subrutina ’imprimir_mensaje’ mostró el menú de manera efectiva, mejorando la interaccion del usuario.

Las subrutinas ejecutaron correctamente los cálculos según la opción seleccionada, con validaciones adecuadas de ’n’ y ’m’, las subrutinas ’solicitar_entero’, ’comprobar_que_es_cero_o_mayor1’, ”comprobar_que_n_mayor_que_m’, ’permutacion’, ’resta’, ’multiplicacion’, ’division’, garantizaron la precisión y validez de los resultados.

Los resultados de combinatoria, permutación y variación se mostraron correctamente en la consola.

V. Conclusiones

En este trabajo se desarrollaron e implementaron dos algoritmos en MIPS: un algoritmo de ordenamiento por selección y un conjunto de operaciones de matemática discreta. El primer algoritmo demostró ser eficaz para organizar hasta 8 elementos ingresados por el usuario, mostrando correctamente el resultado ordenado en la consola.

El menú interactivo para las operaciones de matemática discreta permitió a los usuarios seleccionar y calcular combinatoria, permutación y variación de manera efectiva.

La principal aportación de este trabajo radicó en la aplicación de conceptos de arquitectura de computadores y programación en ensamblador para resolver problemas reales. La utilización del simulador MARS facilitó el desarrollo de los algoritmos, demostrando la importancia de una correcta gestión de la memoria y el uso de subrutinas en programación de bajo nivel.

Referencias

[1] C. G. y Benjamin Sandoval, “Desarrollo de una solución a la asignación de horarios de universitarios

en lenguaje mips," in *CamilaGarate1-BenjaminSandoval-Lab1*, 2024. [Online]. Available: https://es.overleaf.com/read/rhtmmdtwhsjf#8a43d6

[2] Geeksforgeeks, "Selection sort – data structure and algorithm tutorials," in *GeeksforGeeks*, Recuperado 22 de Mayo de 2024. [Online]. Available: https://www.geeksforgeeks.org/selection-sort/

[3] U. de Chile, "Ordenamiento," in *Computacion I*, Recuperado 24 de Mayo de 2024. [Online]. Available: https://users.dcc.uchile.cl/~lmateu/CC10A/Apuntes/ordenamiento/index.html

[4] superprof, "Variaciones, permutaciones y combinaciones," in *Matemática*, Recuperado 22 de Mayo de 2024. [Online]. Available: https://www.superprof.es/apuntes/escolar/matematicas/probabilidades/combinatoria/variaciones-permutaciones-y-combinaciones.html

[5] U. C. I. de Madrid, "Mecanismos de programación: Subrutinas," in *Sistemas Digitales Basados en Microprocesadores*, Recuperado 22 de Mayo de 2024. [Online]. Available: https://ocw.uc3m.es/pluginfile.php/3814/mod_page/content/23/sdbm_tema3_ensamblador_2020_v1.pdf