Informe de Práctica de Laboratorio II

José M. Díaz M. (25.682.785)
Sergio José Noguera (30.572.211)

15 de julio de 2025

1. ¿Qué diferencias existen entre registros temporales ($t0–$t9) y registros guardados ($s0–$s7) y cómo se aplicó esta distinción en la práctica?

Diferencias

- Los registros temporales ($t0-$t9): Se utilizan para valores intermedios en funciones, es decir dan resultados que ayudan a calcular los resultados finales y pueden ser sobrescritos de ser necesario ya que nos tan relevantes sus valores originales.

- Los registros guardados ($s0-$s7): Se utilizan para valores locales importantes o valores que no tengan que cambiar su valor en una función, es decir que sus valores deben ser constantes una vez establecidos y deben ser el mismo tanto al entrar como al salir de la función

Distinción en la práctica

- Los registros temporales ($t0-$t9): Se usaron para valores despreciables, en este caso fueron útiles para los valores de los índices del arreglo de forma que se podían incrementar su valor sin afectar el funcionamiento del algoritmo, también se usaron para contadores para imprimir los arreglos ya que no importaba que se alterara su valor.

- Los registros guardados ($s0-$s7): Se usaron para cargar los valores más importantes de los algoritmos, siendo los números dentro del arreglo y el tamaño del arreglo, estos valores tenían que permanecer sin cambios a lo largo del algoritmo, también se usaron al comenzar el ordenamiento (tanto en el ordenamiento burbuja como el ordenamiento selección) para guardar los valores los valores relevantes, como los contadores.

2. ¿Qué diferencias existen entre los registros $a0–$a3, $v0–$v1, $ra y cómo se aplicó esta distinción en la práctica??

Diferencias

- Los registros $a0-$a3: Se utilizan para guardar o cargar argumentos para ser usados en el algoritmo, sus valores no son resultados sino para declarar parámetros específicos.

- Los registros $v0-$v1: Se utilizan para guardar los valores de retorno de las funciones para ser usados en otra, los valores de estos registros siempre serán únicos.

- El registro $ra: Se utiliza para guardar la dirección de retorno de una función previamente establecida, así al terminar la función retornara a la que lo llamo en primer lugar.

Distinción en la práctica

- Los registros $a0-$a3: Se usaron para cargar las direcciones de los datos, tanto los mensajes, como los números dentro de arreglo se consideran argumentos, así que con estos son cargados para que cuando se haga una llamada al sistema estos estos listos para ser procesados y ser impresos por pantalla.

- Los registros $v0-$v1: Se usaron exclusivamente para hacer llamadas al sistema; es decir, preparar el entorno para imprimir los datos requeridos.

- El registro $ra: Se usó únicamente para retornar a la función que llamo en primer lugar al algoritmo de ordenamiento (tanto en el ordenamiento burbuja como el ordenamiento selección) de forma que se pueda finalizar.

3. ¿Cómo afecta el uso de registros frente a memoria en el rendimiento de los algoritmos de ordenamiento implementados?

La clave del rendimiento radica en la velocidad de acceso a la memoria principal, el cual es mucho más lento que los registros ejecutados por cientos o miles de ciclos de reloj, cada vez que el procesador necesita un dato que no está en un registro, debe ir a la memoria, lo que pausa significativa en la ejecución, en este caso si los arreglos utilizados son muy grandes y no caben completamente en los caches, cada elemento que no esté en el cache implicara entrar a la memoria principal, ralentizando considerablemente el algoritmo. También la cantidad de operaciones que implican a cada registro puede afectar el rendimiento, siendo en este caso que si el arreglo está muy desordenado se tendrán que procesar muchos cambios de valores en los registros a procesar.

4. ¿Qué impacto tiene el uso de estructuras de control (bucles anidados, saltos) en la eficiencia de los algoritmos en MIPS32?

Los bucles anidados:

Estos son conocidos por permitir realizar una misma tarea múltiples veces en un algoritmo siguiendo una lógica basada en N (número de veces que realiza el ciclo), su impacto en la eficiencia recae en su complejidad temporal, cuando se ejecuta un bucle se dice que su complejidad temporal es $\mathcal{O}(N)$ (Número de órdenes que hará el bucle), cuando se implementan bucles anidados éste aumenta exponencialmente su complejidad llegando a $\mathcal{O}(N^2)$, $\mathcal{O}(N^3)$, $\mathcal{O}(N^4)$, y así sucesivamente afectando su eficiencia en altos niveles.

Saltos:

Estos son cocidos por permitir ir de una función a otra, su impacto en la eficiencia recae en el Costo de Cálculo de Dirección, MIPS32 se caracteriza por tener segmentación de instrucciones dividas por funciones y los saltos para llegar a las funciones, calculan su dirección de destino, que depende de su localización el algoritmo siendo que mientras más lejos se encuentre la instrucción de salto de su destino el cálculo tendrá un mayor costo y afectando significativamente la eficiencia.

5. ¿Cuáles son las diferencias de complejidad computacional entre el algoritmo Bubble Sort y el algoritmo alternativo? ¿Qué implicaciones tiene esto para la implementación en un entorno MIPS32?

Los algoritmos Bubble Sort y Selection Sort comparten algunas características fundamentales pero también presentan importantes diferencias que afectan su comportamiento y eficiencia. He aquí una comparación detallada:

CARACTERÍSTICA	BUBBLE SORT	SELECTION SORT
Complejidad temporal mejor caso	$\mathcal{O}(N)$	$\mathcal{O}(N^2)$
Complejidad temporal promedio	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Complejidad temporal peor caso	$\mathcal{O}(N^2)$	$\mathcal{O}(N^2)$
Estabilidad	Estable	No estable
Número de intercambios	Máximo posible	Mínimo necesario
Optimización para datos ordenados	Se puede optimizar con bandera	No se puede optimizar

Implicaciones para Implementación en MIPS32:

Las diferencias en complejidad tienen impactos significativos en la implementación en MIPS32:

1 - Uso de Registros:

- Selection Sort requiere menos registros debido a sus operaciones más simples.
- Bubble Sort necesita más registros para manejar la bandera de optimización y los índices adicionales.

2 - Memoria y Acceso:

- Ambos algoritmos tienen complejidad espacial $\mathcal{O}(1)$.
- Selection Sort realiza menos accesos a memoria por tener menos intercambios.
- Bubble Sort puede beneficiarse de la cache en arrays pequeños debido a su acceso secuencial.

3 - Pipeline y Branch Prediction:

- Selection Sort tiene un patrón de ramificación más predecible.
- Bubble Sort puede beneficiarse de predicción de saltos cuando usa la optimización de bandera.

6. ¿Cuáles son las fases del ciclo de ejecución de instrucciones en la arquitectura MIPS32 (camino de datos)? ¿En qué consisten?

El ciclo de ejecución de instrucciones en la arquitectura MIPS32 se divide en cinco fases fundamentales que forman parte del camino de datos (datapath). Cada fase tiene una función específica y contribuye al proceso de ejecución de las instrucciones. He aquí cada una de estas fases y su función específica:

FASE	FUNCIÓN PRINCIPAL	LATENCIA TÍPICA
Instruction Fetch (IF)	Traer la instrucción desde memoria usando el PC actual	200ps
Instruction Decode (ID)	Decodificar la instrucción y leer los registros fuente	100ps
Execute (EX)	Realizar operaciones ALU o calcular direcciones	200ps
Memory Access (MEM)	Acceder a memoria para lecturas/escrituras	200ps
Write Back (WB)	Guardar resultados en el archivo de registros	100ps

Entre cada fase hay registros especiales que almacenan temporalmente la información mientras fluye por el pipeline. Estos registros son fundamentales porque permiten que diferentes instrucciones estén en diferentes fases simultáneamente:

- Registro IF/ID: Almacena el valor del PC y la instrucción (64 bits).
- Registro ID/EX: Contiene operandos extendidos y señales de control (128 bits).
- Registro EX/MEM: Guarda resultados intermedios y flags (97 bits).
- Registro MEM/WB: Almacena resultados finales y datos leídos (64 bits).

Esta organización en fases permite que múltiples instrucciones estén siendo procesadas simultáneamente, lo que mejora significativamente el rendimiento de la CPU. Por ejemplo, mientras una instrucción está siendo decodificada, otra puede estar trayendo su instrucción de memoria, y otra más puede estar realizando cálculos en la ALU.

7. ¿Qué tipo de instrucciones se usaron predominantemente en la práctica (R, I, J) y por qué?

En la práctica predominaron las instrucciones de tipo I, la razón por que se considera la predomínate se puede explicar mencionando los propósitos de cada una de las instrucciones usadas:

- Para Acceder a la Memoria se utilizaron las instrucciones como lw y sw, esto se debe que se opera directamente sobre un array en memoria y que, en cada iteración, se acceden repetidamente los elementos de los arreglos. Su función más importante fue hacer el intercambio de números en el ordenamiento.

- Para los bucles se utilizaron las instrucciones bge y addiu, los bucles anidados, necesitan dos cosas, un contador que vaya incrementado el valor del índice, este es proporcionado por addiu y una condición de salida que permita para el ciclo para iniciar el siguiente, esto es proporcionado por bge.

- Para las operaciones inmediatas se utilizó las instrucciones addi, las operaciones de direcciones de memoria a menudo implican la adición de un offset inmediato para acceder a elementos específicos dentro de la memoria, en este caso acceder a los números dentro de los arreglos.

8. ¿Cómo se ve afectado el rendimiento si se abusa del uso de instrucciones de salto (j, beq, bne) en lugar de usar estructuras lineales?

El rendimiento en la arquitectura MIPS32 se ve significativamente afectado cuando se abusa de las instrucciones de salto (j, beq, bne) en comparación con estructuras lineales debido a varios factores fundamentales que impactan directamente en el flujo de ejecución y la eficiencia del procesador.

Impacto Principal: Predicción de Saltos

- Las instrucciones de salto rompen el flujo de ejecución secuencial del programa.

- El procesador debe mantener un predictor de saltos para intentar anticipar correctamente la dirección del salto.

- Cuando hay muchos saltos, aumenta la probabilidad de predicciones incorrectas.

- Una predicción errónea resulta en una penalización significativa de rendimiento.

Impacto en el Pipeline

- El pipeline del procesador MIPS32 está optimizado para instrucciones secuenciales.

- Los saltos condicionales (beq, bne) requieren:

1 - AEvaluar la condición.
2 - Calcular la dirección de destino.
3 - Posiblemente rellenar el pipeline nuevamente.

- Cuando hay múltiples saltos cercanos, esto puede causar:

◦ Stalls en el pipeline.
◦ Mayor consumo de energía.
◦ Reducción en el throughput de instrucciones.

Limitaciones Arquitecturales

1 - Rango de Saltos:

- Los saltos condicionales tienen un rango limitado de ±215 palabras.
- Saltos más largos requieren secuencias de múltiples instrucciones.
- Esto aumenta la complejidad y el overhead.

2 - Formato de Instrucciones:

- Las instrucciones de salto utilizan formato I-type.
- Requieren cálculos adicionales para resolver direcciones.
- Mayor complejidad en el decodificador.

Consideraciones Prácticas

- La optimización excesiva puede afectar la mantenibilidad del código.
- Algunos casos requieren inevitablemente el uso de saltos.
- El equilibrio entre rendimiento y claridad del código es crucial.
- Las herramientas modernas de compilación pueden optimizar automáticamente muchas estructuras de control.

Mientras que las instrucciones de salto son fundamentales en la programación MIPS32, su uso excesivo puede tener un impacto significativo negativo en el rendimiento debido a los desafíos de predicción, el manejo del pipeline y las limitaciones arquitecturales. La mejor práctica es utilizarlas solo cuando sean verdaderamente necesarias, prefiriendo estructuras lineales cuando sea posible sin comprometer la funcionalidad o legibilidad del código.

9. ¿Qué ventajas ofrece el modelo RISC de MIPS en la implementación de algoritmos básicos como los de ordenamiento?

El modelo RISC de MIPS ofrece varias ventajas significativas en la implementación de algoritmos de ordenamiento, fundamentadas en sus características arquitectónicas específicas:

Ventajas Principales

1 - Simplicidad y Predecibilidad:

- Instrucciones simples y de ciclo único.
- Pipeline eficiente con menor riesgo de conflictos.
- Facilita la optimización de bucles críticos.

2 - Eficiencia en Memoria:

- Modelo de memoria consistente y predecible.
- Acceso secuencial optimizado para operaciones de comparación.
- Mejor rendimiento en acceso a arrays y estructuras de datos lineales.

3 - Optimización del Pipeline:

- Instrucciones simples permiten mejor flujo en el pipeline.
- Menos estancamientos por dependencias entre instrucciones.
- Mayor aprovechamiento de recursos computacionales.

Ventajas Específicas para Algoritmos de Ordenamiento

1 - Para Algoritmos Iterativos:

- Burbuja: Mayor eficiencia en las iteraciones sucesivas.
- Selección: Optimización en los ciclos de búsqueda.
- Inserción: Mejor manejo de la memoria cache.

2 - Para Algoritmos Recursivos:

- Quicksort: Eficiente gestión del stack.
- Merge Sort: Optimización en la fusión de subarrays.
- Heap Sort: Mejor rendimiento en operaciones aritméticas.

Consideraciones Prácticas

1 - Implementación:

- Código más limpio y mantenible.
- Fácil depuración debido al modelo de memoria consistente.
- Mejor portabilidad entre diferentes implementaciones.

2 - Rendimiento:

- Tiempo de ejecución más predecible.
- Mejor escalabilidad con conjuntos de datos grandes.
- Eficiente uso de recursos computacionales.

La arquitectura RISC de MIPS ofrece ventajas significativas para la implementación de algoritmos de ordenamiento debido a su diseño simple pero potente. La combinación de instrucciones simples, pipeline eficiente y modelo de memoria consistente crea un entorno ideal para optimizar tanto algoritmos iterativos como recursivos. Estas características no solo mejoran el rendimiento sino que también facilitan el desarrollo y mantenimiento del código, lo cual es especialmente valioso en aplicaciones que requieren manipulación intensiva de datos.

10. ¿Cómo se usó el modo de ejecución paso a paso (Step, Step Into) en MARS para verificar la correcta ejecución del algoritmo?

Conceptos Fundamentales

1 - Modo Paso a Paso (Step Over):

- Ejecuta una instrucción completa antes de pausar.
- No entra en las funciones llamadas.
- Ideal para seguir el flujo principal del programa.

2 - Entrar en Paso (Step Into):

- Penetra dentro de las funciones cuando las encuentra.
- Permite examinar el código fuente de las funciones.
- Útil para debuggear funciones específicas.

Fase de Preparación

1 - Configuración inicial:

- Abrir el programa MIPS en MARS.
- Compilar el código para asegurar que no haya errores sintácticos.
- Configurar la ventana de registro pulsando View → CPU Registers.
- Activar la ventana de memoria si es necesario: View → Memory.

2 - Configuración breakpoints:

- Colocar el cursor en la línea donde desea comenzar la depuración.
- Presionar F2 o hacer clic derecho y seleccione "Toggle breakpoint".
- Los breakpoints aparecerán marcados con un punto rojo.

Fase de Ejecución Paso a Paso

1 - Inicio de Depuración:

- Presionar F5 para ejecutar hasta el primer breakpoint.
- El programa se detendrá automáticamente en esa línea.

2 - Uso de Step Into (F7):

- Presionar F7 para ejecutar una instrucción.
- Observar cómo cambian los valores en la ventana de registros.

- Verificar el valor del Program Counter (PC) después de cada paso.

3 - Consideraciones Especiales:

- En las instrucciones branch o jump, observar que se ejecuta automáticamente la instrucción en el slot de delay.
- El registro PC mostrará primero la dirección de la instrucción en el slot de delay.
- Luego avanzará a la dirección de destino del salto.

Fase de Verificación

1 - Monitoreo de Registros:

- Comparar los valores esperados con los reales en cada registro.
- Prestar especial atención a:

◦ Registros $t0-$t9 para datos temporales.
◦ Registros $s0-$s7 para valores salvados.
◦ Registro $v0 para resultados de funciones.
◦ Registro $ra para direcciones de retorno.

2 - Identificación de Errores:

- Si encuentra un valor incorrecto, puede:

◦ Reiniciar la depuración desde el último breakpoint.
◦ Modificar el código si es necesario.
◦ Agregar más breakpoints para mayor precisión.

11. ¿Qué herramienta de MARS fue más útil para observar el contenido de los registros y detectar errores lógicos?

La herramienta más útil en MARS para observar el contenido de los registros y detectar errores lógicos es el panel de Registros (Registers) junto con el Modo Paso a Paso (Single Step Mode). Esta combinación proporciona una visibilidad completa del estado del procesador durante la ejecución del programa.

Panel de Registros

- Se encuentra ubicado en la parte superior derecha de la interfaz de MARS.
- Muestra el valor actual de todos los registros de propósito general ($zero hasta $ra).
- Actualiza automáticamente sus valores durante la ejecución.
- Permite ver cambios en tiempo real mientras se ejecuta el programa.

Modo Paso a Paso

- Accesible mediante F7 o el botón correspondiente en la barra de herramientas.
- Ejecuta una instrucción por vez.
- Facilita el seguimiento detallado de modificaciones en los registros.
- Ideal para verificar operaciones aritméticas y lógicas.

12. ¿Cómo puede visualizarse en MARS el camino de datos para una instrucción tipo R? (por ejemplo: add)

Para visualizar el camino de datos de una instrucción tipo R en MARS, podemos utilizar varias herramientas integradas que nos permiten observar paso a paso cómo fluyen los datos a través del pipeline.

HERRAMIENTAS EN MARS PARA VISUALIZAR EL CAMINO DE DATOS:

Panel de Registros:

- Abrir MARS y escriba una instrucción tipo R simple como add $t0, $t1, $t2.
- Activar la ejecución paso a paso (botón de "single-step").
- Observar en el panel de registros cómo cambian los valores durante cada etapa.

Herramienta de Visualización de Referencias de Memoria:

- Desde el menú Tools, seleccionar "Memory Reference Visualization".
- Conectar la herramienta haciendo clic en Çonnect to MIPS".
- Ejecutar el programa paso a paso para ver el flujo de datos

Configuración de Velocidad de Ejecución:

- Ajustar la velocidad de ejecución a aproximadamente 10 instrucciones por segundo.
- Esto le permitirá observar claramente cada etapa del proceso.

SECUENCIA DE OBSERVACIÓN:

Durante la ejecución de la instrucción ADD, observAR:

1 - La lectura simultánea de los registros fuente ($t1 y $t2).
2 - La operación de suma en la ALU.
3 - El almacenamiento del resultado en el registro destino ($t0).

13. ¿Cómo puede visualizarse en MARS el camino de datos para una instrucción tipo I? (por ejemplo: lw)

Configuración Inicial

- Abrir MARS en la computadora.

- Ir a "File", seguido ir a "New" para crear un archivo .asm, o ve a "Open" y selecciona tu archivo .asm de preferencia que contenga el código MIPS.

- Hacer click en el botón Assemble (el icono de la llave inglesa) para ensamblar tu código.

Acceder a la Vista del Datapath

- Una vez que el programa esté ensamblado, ir al menú Tools de la barra superior.

- Selecciona MIPS para luego ir a "MIPS X-Ray", esto abrirá una nueva ventana que muestra una representación gráfica del datapath MIPS.

Ejecutar la Instrucción lw paso a paso

- En la ventana principal de MARS (donde está el código y los registros), asegúrate de que el contador paso a paso esté apuntando a la instrucción lw que deseas visualizar. Puedes hacer esto en el datapath reiniciando la simulación en "Reset" y luego usando el botón "Run one step at time" para avanzar hasta esa instrucción, así empezara a visualizarse datapath.

- Observa la ventana del Datapath: A medida que ejecutas la instrucción lw paso a paso, verás cómo las señales y los datos fluyen a través de los componentes del datapath. Los caminos activos y los valores en los registros y unidades funcionales se iluminarán o mostrarán sus valores.

14. Justificar la elección del algoritmo alternativo

La elección del algoritmo Selection Sort se debe a su alto rendimiento, el nulo uso de memoria adicional y que no tiene mucha dificultad para ser comprendido, esto se debe ya que cuenta con la característica de ser un algoritmo iterativo, el cual es bien sabido le da una ventaja sobre algoritmos de ordenamiento que usan recursividad, su ventaja principal en este caso es que siempre buscara hacer el menor número de iteraciones posibles.

Hay que recalcar que cuenta con la desventaja de que este rendimiento es fijo según el número de elementos, por lo que siempre tardara lo mismo, también que no es recomendable para arreglos muy grandes debido a baja drásticamente su rendimiento.

15. Análisis y Discusión de los Resultados

Se puede concluir que cada uno de los diferentes registros que existen en MIPS32 resultaron ser útiles para la realización de esta práctica, algunos más usados que otros, pero al final todos cumplen con su propósito de guardar valores o direcciones necesarias para el correcto funcionamiento de los algoritmos de ordenamiento burbuja y el de ordenamiento selección.

Algo a destacar es que para los dos algoritmos realizados, las instrucciones predominantes fueron las de tipo I lo que no fue sorpresa, ya que cuentan con el uso de las estructuras de control (bucles anidados y saltos), que son las responsables de llevar a cabo el ordenamiento.

Otra cosa a relatar fue el aprendizaje de los caminos de datos para distintos tipos de instrucciones que fue posible gracias a la herramienta que ofrece la plataforma MARS donde el datapath puede mostrar todos los caminos posibles.