# Práctica 2: Algoritmos de Ordenación

Rolando Suarez V- 30445947
Jesus Muñoz V- 27188137

## 1. Diferencias entre registros temporales ($t0–$t9) y registros guardados ($s0–$s7)

La diferencia entre los registros temporales ($t0–$t9) y los registros guardados ($s0–$s7) en MIPS32 radica principalmente en su **uso y persistencia de valor durante las llamadas a funciones**.

**Registros temporales ($t0–$t9):** Se utilizan como auxiliares para ciertos cálculos o durante la aplicación de procedimientos. No tienen el objetivo de conservarse durante llamadas a funciones o procedimientos, y su valor puede variar a lo largo de un programa.

**Registros guardados ($s0–$s7):** Por otro lado, son registros cuyo valor busca preservarse durante las llamadas a una o varias funciones. Una función que utilice un registro guardado debe devolverlo a su valor original antes de retornar. Esto se logra típicamente utilizando la pila de memoria, por ejemplo. En la práctica, esto fue precisamente lo que se hizo al utilizar la pila de memoria en funciones con llamadas recursivas.

## 2. Diferencias entre los registros $a0–$a3, $v0–$v1, y $ra

Estos registros cumplen funciones específicas en el contexto de las llamadas a funciones en MIPS32:

- **Registros de argumentos ($a0–$a3):** Se utilizan para pasar **parámetros** a un procedimiento. Son 4 en total, lo que significa que en MIPS32, a priori, podemos declarar e implementar funciones con hasta 4 parámetros.
- **Registros de valor de retorno ($v0–$v1):** Son los registros para los **valores de retorno** de una función. Son 2 registros, aunque, por lo general, en la práctica solo se utiliza uno ($v0).
- **Registro de dirección de retorno ($ra):** Es un solo registro que toma valor automáticamente al hacer el llamado a una función dentro del cuerpo principal de nuestro programa. Guarda el Program Counter de la línea siguiente al llamado de la función. Se usa con la instrucción ‘jr‘ al final de un procedimiento para retornar a esa línea mencionada.

## 3. Impacto del uso de registros frente a memoria en el rendimiento

El uso de **registros** en vez de la **memoria** del computador en cualquier algoritmo aumentará su rendimiento. En el caso de los algoritmos de ordenación, que implican la lectura y escritura en memoria debido al uso de arreglos, la diferencia es notable. Esto es especialmente cierto cuando se comparan algoritmos de implementación más eficiente o de un orden de complejidad BigO menor, ya que los accesos a registros son significativamente más rápidos que los accesos a memoria.

## 4. Impacto de las estructuras de control en la eficiencia en MIPS32

Las estructuras de control tienen un impacto directo en la eficiencia y la complejidad en tiempo de ejecución de los algoritmos.

- **Bucles:** Introducen una nueva dimensión al tiempo de ejecución, el cual ahora dependerá del tamaño del bucle a recorrer. Si un bucle contiene otro bucle anidado, el tiempo de ejecución se calcula al cuadrado. Por ejemplo, el tiempo de ejecución del Bubble Sort es de $O(n^2)$ debido a que tiene un ciclo con otro ciclo anidado.
- **Saltos:** Aunque las instrucciones de salto tienen un tiempo de ejecución bajo en términos de ciclos de reloj, su uso excesivo puede ocasionar problemas de rendimiento debido a interrupciones en el **pipeline del procesador**. El predictor de salto del procesador a veces se ve interrumpido si el salto predicho es incorrecto, lo que resulta en una penalización en el rendimiento.

## 5. Diferencias de complejidad computacional entre Bubble Sort y el algoritmo alternativo

Ambos algoritmos, **Bubble Sort** y el algoritmo alternativo, **Selection Sort**, tienen un orden de complejidad temporal de $O(n^2)$ en el peor caso. Su comportamiento es similar en casos intermedios, siendo el Selection Sort ligeramente más eficiente.

Sin embargo, una diferencia notable en MIPS32 es la cantidad de veces que acceden a memoria para leer y escribir, lo que se evidencia en la cantidad de intercambios:

- **Selection Sort:** En el peor caso, realiza $n$ intercambios.
- **Bubble Sort:** Realiza $n^2$ intercambios.

Esto implica que Bubble Sort accede y escribe en memoria $n$ veces más. Dado que trabajar a nivel de memoria es más lento que trabajar a nivel de registros, Selection Sort, aunque realiza $n^2$ comprobaciones, las efectúa a nivel de registros, lo que le otorga una ventaja práctica en entornos MIPS32.

## 6. Fases del ciclo de ejecución de instrucciones en la arquitectura MIPS32 (camino de datos)

De manera resumida, las fases del ciclo de ejecución son:

1. **Leer la instrucción de la dirección del PC:** El Program Counter (PC) contiene la dirección de la siguiente instrucción a ejecutar.
2. **Aumentar 4 bytes la dirección del PC:** El PC se incrementa para apuntar a la siguiente instrucción, preparándose para el próximo ciclo.
3. **Buscar la instrucción en la memoria de instrucciones:** La instrucción se recupera de la memoria.
4. **Leer los registros necesarios:** Se leen los valores de los registros involucrados en la instrucción desde el banco de registros (hasta 3, dependiendo del tipo de instrucción).
5. **Usar la unidad aritmético-lógica (ALU):** Se realizan las operaciones necesarias con los registros (comparación, suma, asignación, etc.).
6. **Leer de la memoria de datos (si es necesario):** Algunas instrucciones, como las de carga (‘lw‘), requieren leer datos de la memoria.
7. **Escribir el nuevo dato en el banco de registros (si es necesario):** El resultado de la operación o el dato cargado se escribe de vuelta en el banco de registros.

## 7. Tipos de instrucciones usados predominantemente en la práctica

En la implementación de los algoritmos de ordenación, se utilizaron los tres tipos de instrucciones (R, I, J), pero probablemente predominaron las instrucciones de **tipo I**.

- **Tipo I:** Se usaron predominantemente para realizar sumas con valores inmediatos y para operaciones de lectura y escritura en memoria, fundamentales al trabajar con arreglos y contadores.
- **Tipo J:** Se emplearon en los saltos incondicionales y saltos condicionales (‘beq‘, ‘bne‘) que gestionan los bucles de los algoritmos.
- **Tipo R:** Se utilizaron para sumas de asignación entre registros y para los desplazamientos a la izquierda necesarios en el cálculo de direcciones.

La naturaleza repetitiva de estas instrucciones dentro de los bucles de ordenación asegura su uso constante.

## 8. Impacto en el rendimiento por el abuso de instrucciones de salto

Si se abusa del uso de instrucciones de salto (‘j‘, ‘beq‘, ‘bne‘) en lugar de estructuras lineales, el principal problema que surgirá será la **interrupción del pipeline de datos del procesador** debido a discrepancias con la predicción de saltos.

Los procesadores modernos incorporan una **predicción de saltos** para mejorar la eficiencia del pipeline. Sin embargo, si hay una diferencia entre el salto predicho y el salto que realmente se realiza (tras la verificación lógica de la condición), el pipeline puede vaciarse y reiniciarse, lo que se conoce como un ”stall.º ”burbuja”. Un uso excesivo de saltos condicionales aumenta la probabilidad de que ocurran estos errores, disminuyendo el rendimiento general.

## 9. Ventajas del modelo RISC de MIPS en la implementación de algoritmos básicos

El modelo RISC de MIPS ofrece varias ventajas significativas en la implementación de algoritmos básicos como los de ordenamiento:

- **Simplicidad del conjunto de instrucciones:** Facilita la programación en lenguaje ensamblador y la comprensión de la arquitectura.
- **Ejecución en un solo ciclo de reloj:** Idealmente, cada instrucción se ejecuta en un ciclo de reloj, lo que se traduce en una mayor velocidad de ejecución.
- **Gran número de registros de propósito general:** Permite mantener más datos en la CPU, reduciendo la necesidad de accesos a memoria, que son más lentos.
- **Diseño optimizado para pipeline:** La uniformidad de las instrucciones RISC facilita la implementación de un pipeline eficiente, permitiendo que varias instrucciones estén en diferentes etapas de ejecución simultáneamente.

Estas características contribuyen a un alto rendimiento en la ejecución de algoritmos.

# 10. Uso del modo de ejecución paso a paso (Step, Step Into) en MARS

La función de ejecución **paso a paso (Step/Step Into)** en MARS fue crucial para verificar la correcta ejecución de ambos algoritmos. Esto implicó ensamblar el código y luego observar detenidamente los **paneles de Registros y Segmento de Datos**. Con cada instrucción ejecutada individualmente, se pudo confirmar:

- Que los bucles se incrementaban y decrementaban correctamente.
- Que las direcciones de memoria se calculaban bien.
- Que los valores ($s0, $s1) se cargaban y comparaban adecuadamente.
- Que los intercambios en memoria eran correctos.

Este método permitió una depuración precisa y aseguró que los algoritmos funcionaban como se esperaba.

# 11. Herramienta de MARS más útil para observar registros y detectar errores lógicos

En MARS, la herramienta más útil para observar el contenido de los registros y detectar errores lógicos fue el **Panel de Registros**, ubicado a la derecha de la interfaz. Al combinar su visualización en tiempo real con la ejecución paso a paso (Step/Step Into), se pudo verificar:

- La correcta carga de datos.
- El flujo de los contadores de los bucles.
- Los resultados de las operaciones.
- El comportamiento de las condiciones de salto.

Esto fue indispensable para asegurar que los algoritmos funcionaban exactamente como se esperaba y para identificar rápidamente cualquier discrepancia lógica.

# 12. Visualización del camino de datos en MARS

## 12.1. Para una instrucción tipo R (por ejemplo: `add`)

Para visualizar el camino de datos de una instrucción de tipo R en MARS:

1. Realizar la ejecución paso a paso del algoritmo hasta llegar a una instrucción de tipo R (ej., ‘add‘).
2. Dirigirse a la barra de herramientas superior y seleccionar **”Tools”**.
3. Dentro de ”Tools”, seleccionar **”MIPS X-RAY”**.
4. En la nueva ventana que aparece, seleccionar **Çonnect to MIPS”**.

El simulador mostrará de forma dinámica el camino de datos para la instrucción de tipo R deseada.

### 12.2. Para una instrucción tipo I (por ejemplo: `lw`)

El procedimiento para visualizar el camino de datos de una instrucción de tipo I en MARS es idéntico al de tipo R:

1. Ejecutar el algoritmo paso a paso hasta llegar a una instrucción del tipo deseado (ej., 'lw').
2. Ir a la barra de herramientas superior y seleccionar **"Tools"**.
3. Dentro de "Tools", seleccionar **"MIPS X-RAY"**.
4. En la nueva ventana, presionar **Çonnect to MIPS"**.

De esta manera, aparecerá el camino de datos de forma dinámica para la instrucción de tipo I.

## 13. Justificación de la elección del algoritmo alternativo

Se escogió el algoritmo de ordenamiento **Selection Sort (ordenamiento por selección)** debido a su **simplicidad y lógica intuitiva**. La lógica de este algoritmo es bastante directa: en cada iteración, se busca el elemento más pequeño del arreglo restante y se coloca en su posición final al principio del arreglo, replicando cómo se ordena algo manualmente.

Aunque este algoritmo tiene una complejidad temporal de $O(n^2)$, lo que lo hace inviable para ordenar grandes volúmenes de datos, es ideal para implementaciones en arreglos pequeños o con fines educativos debido a su clara operación paso a paso.

## 14. Análisis y Discusión de los Resultados

Al analizar los resultados entre **Bubble Sort** y **Selection Sort**, ambos son algoritmos de ordenamiento simples con una complejidad de tiempo de $O(n^2)$, haciéndolos inadecuados para grandes volúmenes de datos. La diferencia clave radica en el **número de intercambios**:

- **Bubble Sort:** Puede realizar muchos intercambios ($O(n^2)$), siendo un algoritmo estable y potencialmente más rápido si los datos ya están casi ordenados. Sin embargo, los numerosos intercambios implican frecuentes y costosas operaciones de escritura en memoria.
- **Selection Sort:** Realiza un número mínimo de intercambios (solo uno por iteración principal), lo cual es ventajoso cuando las operaciones de escritura en memoria son costosas. Aunque es inherentemente inestable y su rendimiento es consistentemente $O(n^2)$ independientemente del orden inicial, su eficiencia en la escritura lo hace práctico en ciertos contextos.

En resumen, si bien comparten la misma complejidad teórica, Selection Sort tiende a ser más eficiente en un entorno MIPS32 debido a la menor cantidad de operaciones de escritura en memoria, a pesar de que ambos son ineficientes para tareas de ordenamiento a gran escala.