Informe Practica 2

Samuel Primera C.I: 31.129.684, Samuel Reyna CI: 30.210.759

15 de Julio 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:

- **Registros temporales (`$t0-$t9`):** El procedimiento llamador debe guardarlos en la pila antes de invocar una subrutina, pues el invocado puede modificarlos.
- **Registros guardados (`$s0-$s7`):** El procedimiento invocado debe preservarlos: los guarda al inicio y los restaura antes de retornar.

Aplicación práctica:

- En **bubble sort**:
  - `$s0` y `$s1` almacenan tamaño del arreglo y dirección base (valores persistentes).
  - Guarda/restaura `$s0-$s3` y `$ra` al invocar. `swap`
- En **insertion sort**:
  - `$t0`, `$t1` para índices (`i`, `j`) o valores temporales
  - No se guardan antes de llamadas: responsabilidad del llamador.

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

- **Registros de argumentos (`$a0`–`$a3`)**
  - Pasan parámetros de un procedimiento llamador a uno invocado (hasta 4 parámetros).
  - Parámetros adicionales se almacenan en la pila.
- **Registros de retorno (`$v0`–`$v1`)**
  - Devuelven valores desde el procedimiento invocado al llamador.
- **Registro de dirección de retorno (`$ra`)**
  - Almacena la dirección de retorno tras una llamada a procedimiento.
  - Se usa `jr $ra` para regresar al punto de llamada.

Aplicación práctica (ejemplo en `Burbuja.txt`)

- `$v0`:
  - Gestiona llamadas al sistema: entrada (`li $v0, 5`), impresión de cadenas (`li $v0, 4`) y terminación del programa (`li $v0, 10`).
- `$a0`–`$a1`:
  - Transferencia de parámetros en procedimientos (ej: `sort` recibe `v` y `n` en `$a0`/`$a1`; `swap` usa los mismos para `v` y `j`).
- `$ra`:
  - Se guarda en la pila al inicio de `sort` y se restaura antes de retornar.
  - Permite retornos anidados correctos (`sort` → `swap` → `sort` → llamador original mediante `jr $ra`).

yo quite mi pila

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

Ventajas del uso de registros

- **Máxima velocidad**: Los registros son el recurso más rápido para almacenar datos
- **Acceso eficiente**: MIPS32 cuenta con 32 registros, permitiendo mantener variables críticas cerca de la CPU.
- **Reducción de accesos**: Minimiza operaciones lentas al priorizar datos frecuentes.

Desventajas del uso de memoria

- **Lentitud inherente**: La memoria principal es significativamente más lenta que los registros
- **Riesgos en el pipeline**: Accesos causan *stalls* por *load-use hazards*.
- **Fallos de caché críticos**: Datos ausentes obligan a acceder a niveles más lentos.
- **Operaciones inevitables**: Algoritmos requieren accesos constantes mediante `lw`/`sw`.

Implicaciones para algoritmos de ordenamiento

- **Limitación práctica**:
  - Variables activas en registros, pero **datos masivos en memoria**.
- **Importancia de la jerarquía**:
  - El rendimiento depende del *patrón de acceso* más que del conteo de instrucciones.

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

Los ciclos anidados influyen principalmente en el tiempo de ejecución de un algoritmo. Dependiendo de cómo se implementen, pueden aumentar significativamente el orden de complejidad computacional (por ejemplo, transformando una complejidad lineal O(n) en cuadrática O(n^2) o incluso exponencial). Esto ocasiona múltiples operaciones redundantes de lectura y sobreescritura en los elementos del arreglo, ya sea de forma parcial o total. Como consecuencia, se generan cuellos de botella en el acceso a memoria, uso excesivo de recursos del sistema y lentitud generalizada para completar la tarea encomendada, especialmente con volúmenes grandes de datos.

5. ¿Cuáles son las diferencias de complejidad computacional entre el algoritmo Bubble Sort e Insertion Sort? ¿Qué implicaciones tiene esto para MIPS32?

Complejidad computacional:

Algoritmo	Peor caso	Mejor caso	Caso promedio
Bubble Sort	$O(n^2)$	$O(n)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n)$	$O(n^2)$

Implicaciones en MIPS32:

- **Rendimiento con n grande**:
  - $n = 100k \Rightarrow \approx 10^{10}$ operaciones.
- **Cuellos de botella**:
  - Accesos a memoria dominan tiempo de ejecución.
  - Bucles anidados exacerban riesgos de control.

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 en MIPS32 se divide en **cinco etapas** en una implementación segmentada:

1. **Búsqueda de Instrucción (IF)**:
   - Lee la instrucción de memoria usando la dirección del *Program Counter (PC)*.
   - Incrementa el PC en 4 (siguiente instrucción).
   - Almacena la instrucción y PC+4 en el registro*IF/ID*.
2. **Decodificación y Lectura de Registros (ID)**:
   - Decodifica la instrucción y lee registros fuente del banco de registros.
   - Extiende inmediatos con signo (si aplica).
   - Transmite datos al registro *ID/EX*.
3. **Ejecución (EX)**:
   - La ALU realiza operaciones:
     - Aritméticas/lógicas (ej: `add`, `sub`).
     - Cálculo de direcciones (ej: `lw $t0, 4($s1)`).

    - ◇ Comparaciones para saltos (ej: `beq`, `bne`).
  - Resultados se guardan en *EX/MEM*.
4. **Acceso a Memoria (MEM)**:
  - Accede a memoria solo para `lw` (carga) o `sw` (almacenamiento).
  - Instrucciones no relacionadas con memoria pasan directamente a *MEM/WB*.
5. **Escritura de Resultado (WB)**:
  - Escribe resultados en el banco de registros:
    - ◇ Desde la ALU (instrucciones tipo R).
    - ◇ Desde memoria (instrucciones `lw`).

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

En algoritmos como **bubble sort** e **insertion sort**, predominan:

- **Tipo I (Inmediatas)**:
  - ◇ `lw`/`sw`: Acceso a elementos del array en memoria (ej: `lw $t0, 0($s1)`).
  - ◇ `addi`/`li`: Manipulación de constantes (contadores, índices).
  - ◇ `beq`/`bne`: Bifurcaciones para bucles y condicionales.
- **Tipo R (Registros)**:
  - ◇ `add`/`sub`/`sll`: Operaciones aritméticas y comparaciones (ej: `sll $t2, $t0, $t1`).
- **Tipo J (Saltos)**:
  - ◇ `j`/`jal`: Menos frecuentes (saltos incondicionales o llamadas a funciones como `swap`).

Los algoritmos de ordenamiento requieren acceso constante a arrays (`lw`/`sw`), manipulación de índices (`addi`), y bifurcaciones para bucles (`beq`/`bne`). Las operaciones aritméticas (`add`, `sll`) son esenciales para comparaciones e intercambios.

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?

- **Riesgos de control en el pipeline**:
  - ◇ Los saltos obligan a detener el pipeline (*paradas de la tubería o pipeline stalls*) hasta resolver la dirección destino.
  - ◇ Se insertan "burbujas"(ciclos inactivos), desperdiciando recursos.
- **Penalización por predicción incorrecta**:
  - ◇ Si falla la predicción de saltos, se descartan instrucciones ya procesadas (mayor penalización en pipelines profundos).
- **Estructuras lineales vs. Saltos**:
  - ◇ Código lineal minimiza saltos, maximizando paralelismo (*ILP*) y evitando paradas
  - ◇ Técnicas como *loop unrolling* (o desenrollado de bucles) reducen saltos en bucles, optimizando el flujo

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

- **Simplicidad y regularidad**:

- ◇ Instrucciones de 32 bits fijas y formato uniforme (3 operandos) facilitan la decodificación
- **Pipeline eficiente**:
  - ◇ Etapas balanceadas permiten alta frecuencia de reloj y paralelismo
- **Arquitectura load/store**:
  - ◇ Separación clara entre operaciones (ALU) y acceso a memoria (`lw`/`sw`)
- **32 registros de propósito general**:
  - ◇ Variables frecuentes (índices, contadores) se mantienen en registros, reduciendo accesos a memoria
- **Optimización de compiladores**:
  - ◇ Facilita técnicas como *loop unrolling*(desenrollado de bucles) o reordenamiento de instrucciones

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

Funcionalidad:

- **Step**: Ejecuta una instrucción, ignorando detalles de funciones llamadas (ej: `jal swap`).
- **Step Into**: Ingresa a funciones para depurar internamente (ej: instrucciones dentro de `swap`).

Verificación en algoritmos:

1. **Flujo de control**:
   - Confirma que bucles y bifurcaciones se ejecuten correctamente (ej: `beq` en *bubble sort*).
2. **Estado del procesador**:
   - Monitorea registros (ej: `$t0` para índices, `$s0` para dirección del array)
   - Verifica valores en memoria después de `lw`/`sw` (intercambios en *insertion sort*).
3. **Detección de errores lógicos**:
   - Identifica comparaciones incorrectas (`sll`) o intercambios inválidos.

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

1. **Ventana de Registros (Register Window)**
   - **Acceso**: Menú Tools ¿Registers (panel derecho).
   - **Funcionalidad**:
     - ◇ Muestra en tiempo real los 32 registros de propósito general.
     - ◇ Valores actualizados en hexadecimal/decimal.
     - ◇ Edición manual directa.
   - **Depuración**:
     - ◇ Detección de registros no inicializados.
     - ◇ Identificación de sobrescrituras accidentales.
     - ◇ Verificación de valores intermedios.
2. **Ejecución Paso a Paso (Single Stepping)**
   - **Controles**:
     - ◇ Step: Avance instrucción por instrucción.
     - ◇ Backstep (v4.5+): Retroceso de instrucciones.

- **Aplicación**:
  - Rastreo de saltos erróneos (beq, bne).
  - Detección de bucles infinitos.
  - Análisis de flujo de control.

3. **Puntos de Interrupción (Breakpoints)**
   - **Implementación**: Clic en margen izquierdo del editor.
   - **Ventajas**:
     - Pausa ejecución en líneas críticas.
     - Depuración selectiva de funciones/bucles.
     - Combinación con ventana de registros.
4. **Ventana de Memoria (Data Segment)**.
   - **Acceso**: Tools ¿Data Segment.
   - **Capacidades**:
     - Visualización de arreglos/variables globales.
     - Edición hexadecimal/decimal/ASCII.
     - Monitoreo de reserva de espacio (.data, .text).
   - **Errores detectables**:
     - Desbordamientos de buffers.
     - Accesos a memoria inválidos (lw/sw).

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

1. Abre MARS y carga tu código (ej: `add $t0,$t0,$t1`).
2. Ve al menú `Tools` → `MIPS X-Ray`.
   - Se abrirá una ventana nueva con el diagrama del datapath completo.
3. Ejecuta paso a paso:
   - Usa el botón `Step` (o `F7`) para avanzar instrucción por instrucción.
   - `MIPS X-Ray` resaltará en rojo las rutas y componentes activos durante cada ciclo de reloj.

MARS resalta componentes (ALU, banco de registros) y registros intermedios (ID/EX, EX/MEM) durante la ejecución.

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

1. Abre MARS y carga tu código (ej: `lw $a0, 0($s1)`).
2. Ve al menú `Tools` → `MIPS X-Ray`.
   - Se abrirá una ventana nueva con el diagrama del datapath completo.
3. Ejecuta paso a paso:
   - Usa el botón `Step` (o `F7`) para avanzar instrucción por instrucción.
   - `MIPS X-Ray` resaltará en rojo las rutas y componentes activos durante cada ciclo de reloj.

14. Justificar la elección del algoritmo alternativo

Principalmente por su forma de ejecución. Aunque en promedio ambos tienen un orden de complejidad $O(n^2)$, el Insertion Sort inserta cada elemento en su posición correcta directamente,

mientras que el algoritmo Bubble Sort debe comparar y posiblemente intercambiar elementos uno por uno.

Aunque en casos muy particulares puedan parecer similares, el Insertion Sort resulta una opción más eficiente que el Bubble Sort. Otro punto a su favor es su legibilidad: a pesar de ser un algoritmo un poco más complejo que el Bubble Sort, su funcionamiento es fácil de entender, lo que facilita su implementación para el ordenamiento de arreglos.

15. Análisis y Discusión de los Resultados

Tanto leer los datos de entrada como imprimirlos en ambos algoritmos es similar. Ambos toman el tamaño del arreglo y ejecutan un ciclo para leer los valores del mismo, y otro para imprimirlos. La diferencia radica en la función de ordenamiento.

El algoritmo Bubble Sort necesita dos ciclos: se inicializa el índice superior en $n-1$ para evitar intercambiar elementos que se salgan del rango del vector, y la bandera de intercambios también se inicializa. Luego, se entra en el primer bucle, reiniciando el contador en 0 y la bandera de intercambios. Después, se entra en el segundo bucle, el cual verifica y actualiza el contador y realiza el intercambio de elementos de ser necesario. Esto último se verifica en la etiqueta `end_inner`, que se encarga de revisar si hubo un intercambio. De ser así, se retorna al ciclo exterior para reiniciar la bandera y el contador, ya que no son necesarios más intercambios. En caso contrario, se disminuye el rango y se devuelve al ciclo exterior para continuar. Al terminar de ordenar los elementos, se retorna a la función principal para imprimir los elementos ya ordenados.

El Insertion Sort, al igual que el burbuja, necesita dos ciclos. Inicializa sus pasos (*steps*) en 1 y entra en el primer ciclo. Se guarda el dato a insertar en el arreglo y entra al segundo ciclo, donde se van moviendo de lugar los elementos necesarios para insertar el elemento en la posición correcta. Esto se repite hasta que todos los elementos estén en sus respectivos lugares. Finalmente, se retorna a la función principal para imprimir los elementos ya ordenados.

En conclusión, aunque ambos algoritmos realizan la misma tarea, se puede ver una clara diferencia en su ejecución. También cabe destacar que, en cuanto a eficiencia, si bien ambos tienen la misma complejidad $O(n^2)$, el Insertion Sort es ligeramente más rápido, ya que no tiene que detenerse constantemente a verificar si el elemento es intercambiable.