Informe de Laboratorio 2: Arquitectura MIPS32

Mike Gonzalez y Miguel Monsalves

14 de julio de 2025

Respuestas a las Preguntas

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

- **Registros Temporales ($t0–$t9)**
  - *No preservados (Caller-saved)*: Modificables por subrutinas sin restauración
  - *Responsabilidad del Caller*: Guardar valores antes de llamadas si son necesarios posteriormente
  - *Uso*: Valores temporales dentro de una función
- **Registros Guardados ($s0–$s7)**
  - *Preservados (Callee-saved)*: Subrutinas deben restaurar valores originales
  - *Responsabilidad del Callee*: Guardar/restaurar al inicio/fin de ejecución
  - *Uso*: Variables persistentes entre llamadas a funciones
- **Aplicación práctica**:
  - `$tX` para índices temporales en bucles
  - `$sX` para punteros a arreglos y variables críticas

2. Diferencias entre $a0–$a3, $v0–$v1, $ra

- **Registros de Argumentos ($a0–$a3)**:
  - Paso de parámetros a subrutinas
  - Caller-saved
- **Registros de Retorno ($v0–$v1)**:
  - Devolución de resultados

- Caller-saved

- **Registro $ra**:
  - Almacena dirección de retorno
  - Callee-saved (excepto funciones hoja)
  - Crítico para gestión de llamadas anidadas

3. Uso de registros vs memoria en rendimiento

- **Jerarquía de memoria**:

Nivel	Acceso	Tamaño
Registros	1 ciclo	32 B
Caché L1	3-5 ciclos	32 KB
RAM	100+ ciclos	GBs

- **Impacto en algoritmos**:
  - Operaciones en registros → máximos 1 ciclo
  - Accesos a memoria → penalización por latencia
  - *Bubble Sort*: $O(n^2)$ accesos memoria (ineficiente)
  - *Selection Sort*: Minimiza escrituras (solo $O(n)$)

4. Impacto de estructuras de control

- **Penalizaciones por saltos**:

$$\text{Tiempo}_{\text{ejecución}} = \text{Ciclos} \times \text{CPI} + \text{Penalización}_{\text{saltos}}$$

- **Burbujas en pipeline**: **Ejemplo con instrucción de salto (beq)**:

Ciclo	IF	ID	EX	MEM	WB
1	beq	-	-	-	-
2	instr2	beq	-	-	-
3	instr3	instr2	beq	-	-
4	burbuja	instr3	instr2	beq	-
5	destino	burbuja	instr3	instr2	beq

  Cada burbuja introduce 1 ciclo de retraso (stall) en el pipeline

- **Optimizaciones**:
  - Reducción de saltos redundantes
  - Desenrollado de bucles (loop unrolling)
  - Uso de predicción estática de saltos

5. Complejidad computacional: Bubble Sort vs Selection Sort

Algoritmo	Mejor Caso	Peor Caso	Intercambios
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$	$O(n)$

Implicaciones MIPS32:

- Selection Sort → menos accesos memoria (óptimo)
- Bubble Sort → ventaja en datos casi ordenados

6. Ciclo de ejecución MIPS32

1. **IF (Instruction Fetch)**:
   - Fetch desde memoria de instrucciones
   - PC ← PC + 4
2. **ID (Instruction Decode)**:
   - Decodificación de opcode
   - Lectura de registros
   - Extensión de signo
3. **EX (Execute)**:
   - Operaciones ALU
   - Cálculo de direcciones
4. **MEM (Memory Access)**:
   - Acceso a memoria de datos
5. **WB (Write Back)**:
   - Escritura en banco de registros

7. Tipos de instrucciones predominantes

Tipo	Ejemplos	Uso
I	`lw`, `sw`, `addi`, `beq`	65 %
R	`add`, `sub`, `slt`	30 %
J	`j`, `jal`	5 %

Justificación: Alta frecuencia de accesos a memoria (I) y operaciones aritméticas (R)

Investigación: Preguntas 8-15

8. Abuso de instrucciones de salto

- **Problema**:
  - Burbujas en pipeline (3-5 ciclos perdidos por salto)
  - Ejemplo en Bubble Sort base:

```
bgt $t3, $t4, swap   # Salto condicional
j no_swap            # Salto redundante (flujo natural)
```

- **Solución**:
  - Uso de contador lineal (`$t5`) en Bubble Sort optimizado
  - Reducción de 450 a 383 ciclos (15 % menos)

9. Ventajas modelo RISC/MIPS

- **RISC**: Instrucciones simples + ejecución single-cycle
- **CISC**: Instrucciones complejas + multi-ciclo
- **Ventajas**:
  - Swap en 3 instrucciones: `lw` → `add` → `sw`
  - Pipeline predecible (sin microcódigo)

10. Uso modo paso a paso en MARS

- **Step Into**: Verificación elemento máximo en Bubble Sort
- **Error detectado**: Offset mal calculado en `lw`

```
lw $t3, 8($a0)   # Error: debería ser 4($a0)
```

11. Herramientas de depuración MARS

- **Data Segment**: Detectó que `$a0` se sobrescribía
- **Solución**: Preservar `$a0` en pila

```
addi $sp, $sp, -4
sw $a0, 0($sp)
# ... llamada a función
lw $a0, 0($sp)
addi $sp, $sp, 4
```

12. Camino de datos instrucción tipo R (add)

Etapas para `add $t0, $t1, $t2`:

1. **IF**: Carga la instrucción desde memoria
2. **ID**:
   - Decodifica instrucción (opcode = 000000)
   - Lee registros $t1 y $t2
   - Identifica función ALU (add = 100000)
3. **EX**: ALU realiza suma ($t1 + $t2)
4. **MEM**: Sin acceso a memoria
5. **WB**: Escribe resultado en $t0

Recursos utilizados:

Etapa	Componentes
IF	Memoria Instrucciones, PC
ID	Banco de Registros, Unidad Control
EX	ALU
MEM	-
WB	Banco de Registros

13. Camino de datos instrucción tipo I (lw)

Etapas para `lw $t0, 4($a0)`:

1. **IF**: Carga la instrucción desde memoria
2. **ID**:
   - Decodifica instrucción (opcode = 100011)
   - Lee registro $a0
   - Extiende signo del inmediato (4)
3. **EX**: Calcula dirección ($a0 + 4)
4. **MEM**: Lee dato de memoria en dirección calculada
5. **WB**: Escribe valor en $t0

Recursos utilizados:

Etapa	Componentes
IF	Memoria Instrucciones, PC
ID	Banco de Registros, Extensor Signo
EX	ALU (suma)
MEM	Memoria Datos
WB	Banco de Registros

14. Justificación algoritmo alternativo (Selection Sort)

- **Menos intercambios**: O(n) vs O(n^2)
- **Eficiencia memoria**: Solo 1 swap/iteración
- **Adaptación MIPS**: Lógica intuitiva para depuración

15. Análisis de resultados

Algoritmo (versión)	**Comparaciones**	**Intercambios**	**Ciclos (n=10)**
Bubble Sort (base)	O(n^2)	O(n^2)	450
Bubble Sort (opt.)	O(n^2)	O(n^2)	383
Selection Sort (base)	O(n^2)	O(n)	300
Selection Sort (opt.)	O(n^2)	O(n)	270

Conclusión:

- Selection Sort 40 % más rápido que Bubble Sort
- Optimizaciones redujeron ciclos en 15-30 %
- Menos accesos a memoria = mejor rendimiento

Modalidad de Entrega

1. **Archivos**:
   - Códigos MIPS: 4 archivos .asm (base y optimizados)
   - Informe: `informe.tex` y `informe.pdf`
2. **Defensa presencial**: 15 de julio de 2025
   - Demostración en MARS de versiones optimizadas
   - Análisis de reducción de stalls en pipeline