Informe de Laboratorio: Estructura de Computadores

Nombre del Estudiante: Manuel Cardona Martinez
Fecha: 26/02/2026
Asignatura: Estructura de Computadores

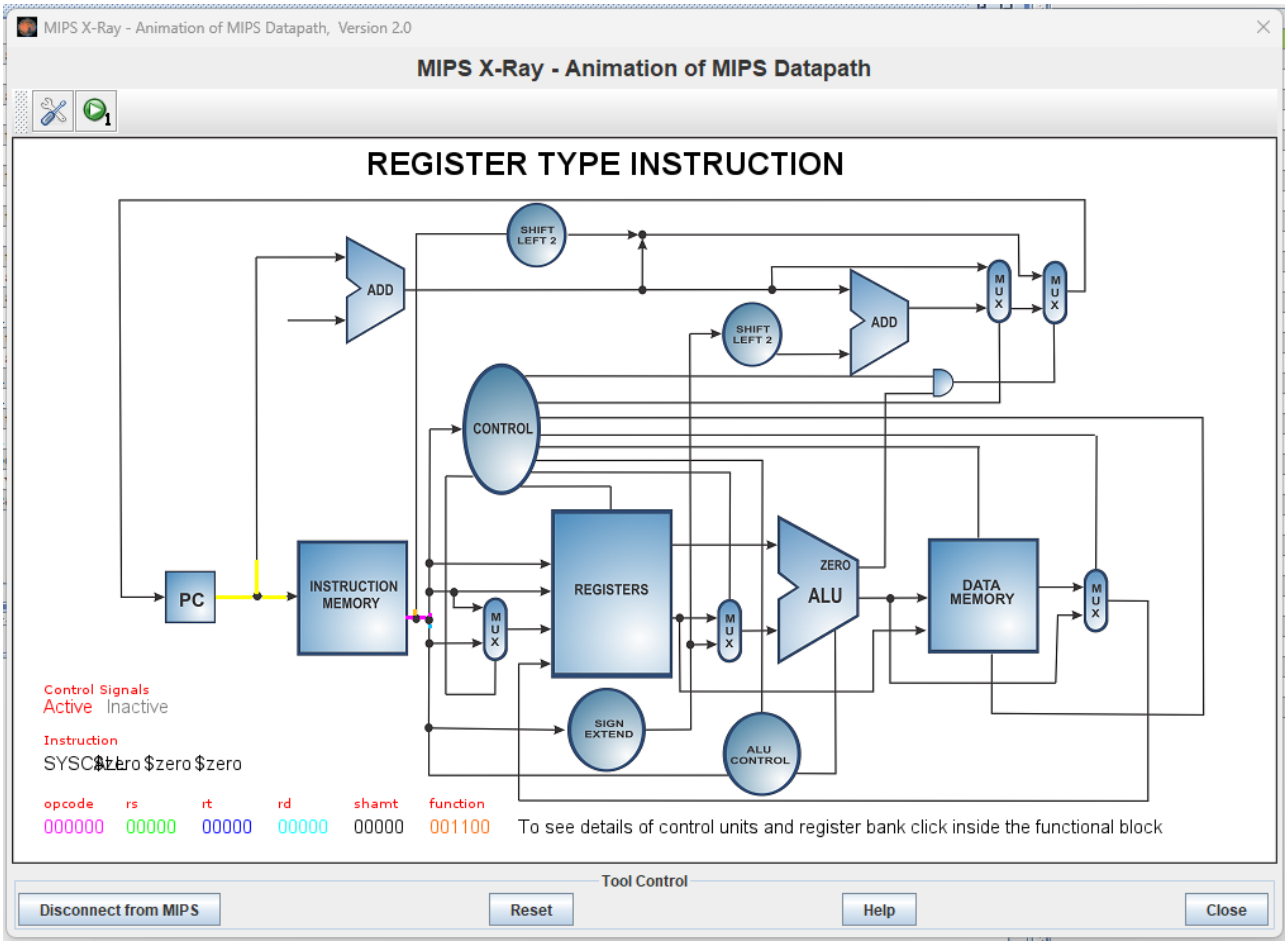
Enlace del repositorio en GitHub: https://github.com/manuel623/Laboratorio1_Estructura_Computadores.git

1. Análisis del Código Base

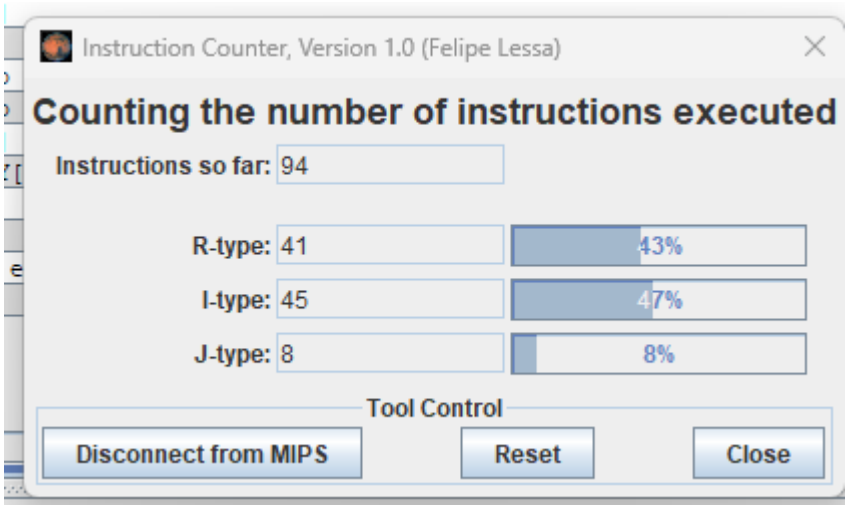
1.1. Evidencia de Ejecución

Adjunte aquí las capturas de pantalla de la ejecución del `programa_base.asm` utilizando las siguientes herramientas de MARS:

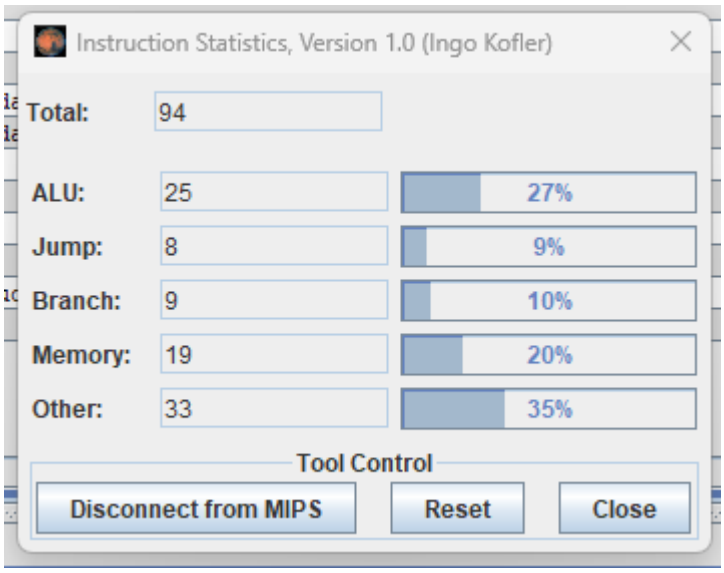
- **MIPS X-Ray** (Ventana con el Datapath animado).



- **Instruction Counter** (Contador de instrucciones totales).



- **Instruction Statistics** (Desglose por tipo de instrucción).



1.2. Identificación de Riesgos (Hazards)

Completa la siguiente tabla identificando las instrucciones que causan paradas en el pipeline:

Instrucción Causante	Instrucción Afectada	Tipo de Riesgo (Load-Use, etc.)	Ciclos de Parada
`lw $t6, 0($t5)`	`mul $t7, $t6, $t0`	Load-Use	1
`mul $t7, $t6, $t0`	`addu $t8, $t7, $t1`	Data-Hazard	0

1.2. Estadísticas y Análisis Teórico

Dado que MARS es un simulador funcional, el número de instrucciones ejecutadas será igual en ambas versiones. Sin embargo, en un procesador real, el tiempo de ejecución (ciclos) varía. Completa la siguiente tabla de análisis teórico:

Métrica	Código Base	Código Optimizado
Instrucciones Totales (según MARS)	94	94
Stalls (Paradas) por iteración	2 (1 Load-Use + 1 Salto)	1 (Salto)

Métrica	Código Base	Código Optimizado
Total de Stalls (8 iteraciones)	16	8
Ciclos Totales Estimados (Inst + Stalls)	110	102
CPI Estimado (Ciclos / Inst)	1.17	1.08

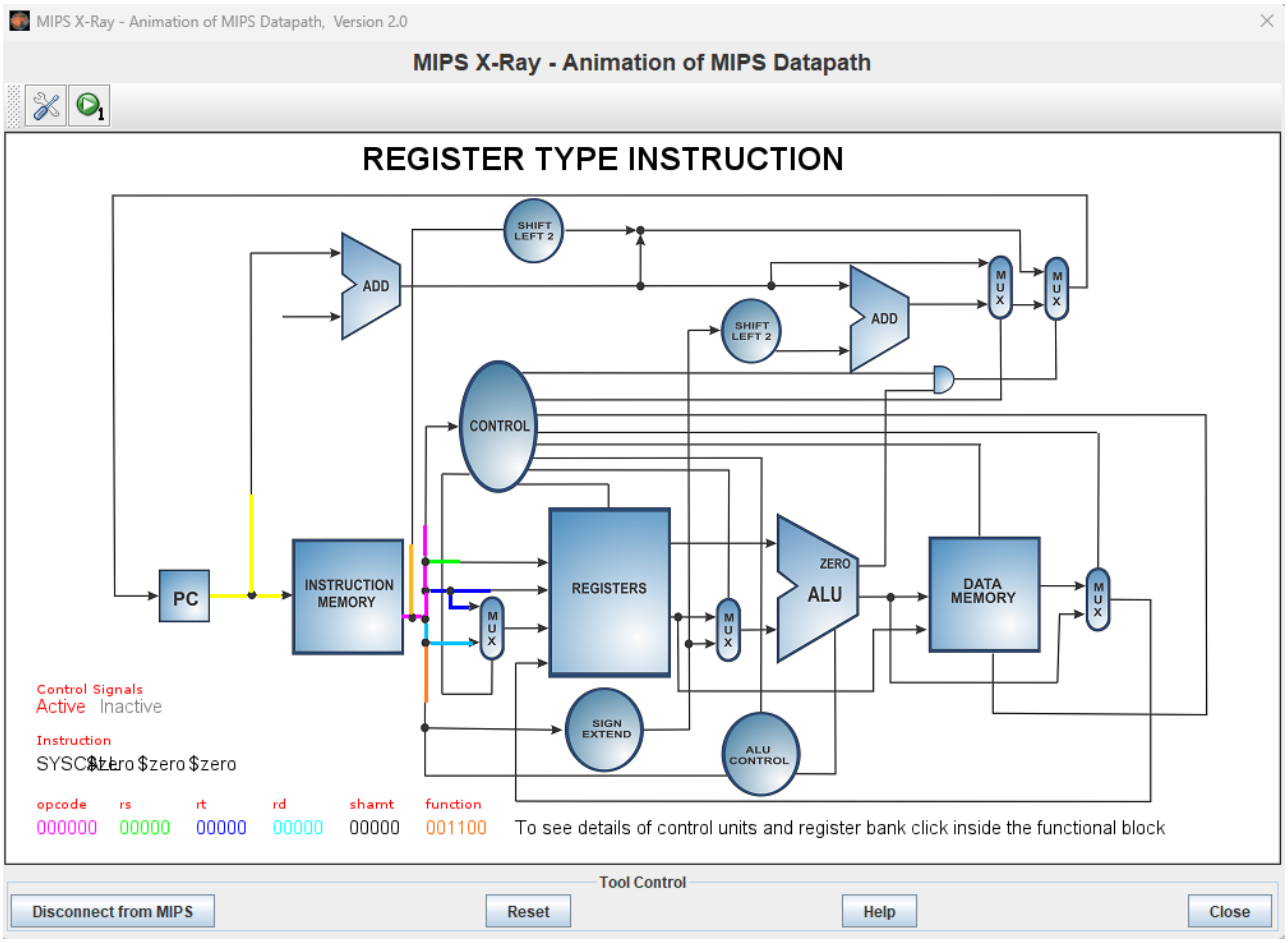
En el total de Stalls del Código base es 16 ya que MARS solo muestra las ejecuciones pero no los retrasos de hardware, en el Código base hay 2 paradas técnicas, la cual es el Load-Use entre lw $t6, 0($t5) y mul $t7, $t6, $t0 y el control hazard que es al finalizar cada loop

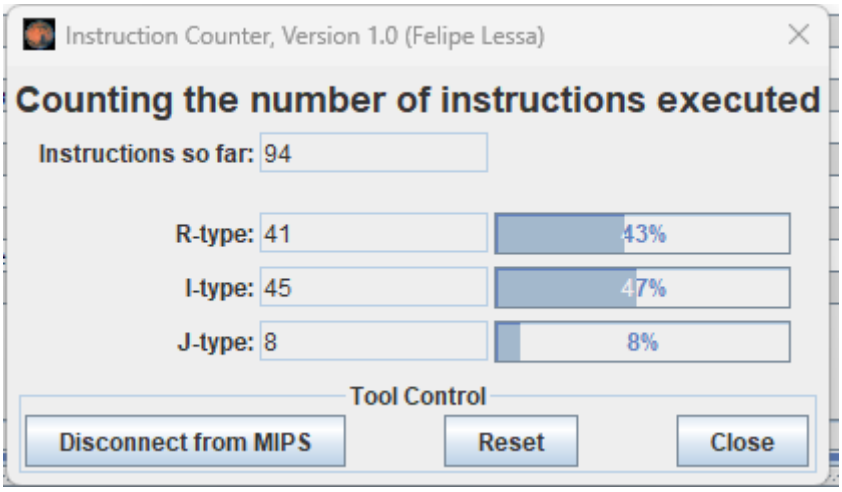
2. Optimización Propuesta

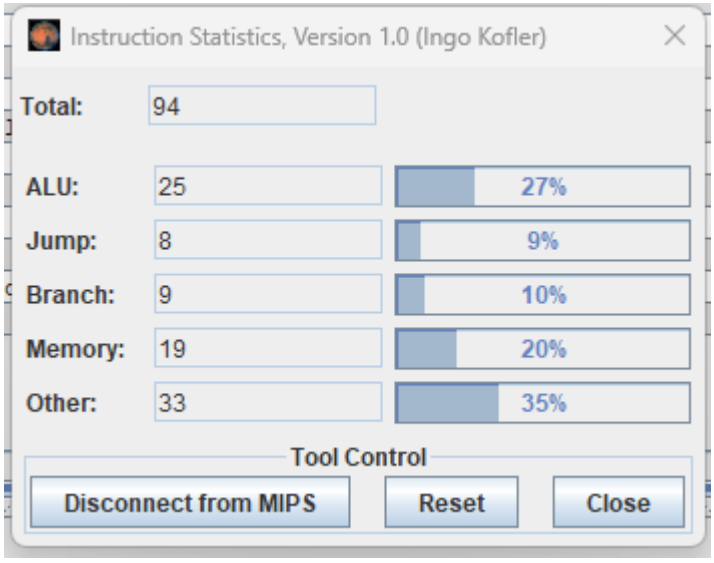
2.1. Evidencia de Ejecución (Código Optimizado)

Adjunte aquí las capturas de pantalla de la ejecución del `programa_optimizado.asm` utilizando las mismas herramientas que en el punto 1.1:

- **MIPS X-Ray**.



- **Instruction Counter**. 

- **Instruction Statistics**. 

2.2. Código Optimizado

Pega aquí el fragmento de tu bucle `loop` reordenado:

```
loop:
    # --- Condición de salida ---
    beq $t3, $t2, fin     # Si i == tamano, salir del bucle

    # --- Cálculo de dirección de memoria ---
    sll $t4, $t3, 2
    addu $t5, $s0, $t4
    addu $t9, $s1, $t4  #se adelanta el calculo

    # --- Carga de dato ---
    lw $t6, 0($t5)
    addi $t3, $t3, 1    # increma i mientras se busca el dato

    # --- Operación aritmética ---
    mul $t7, $t6, $t0
    addu $t8, $t7, $t1

    # --- Almacenamiento de resultado ---
    sw $t8, 0($t9)
    j loop
```

```
fin:
```

2.2. Justificación Técnica de la Mejora

Explica qué instrucción moviste y por qué colocarla entre el `lw` y el `mul` elimina el riesgo de datos:

> En el código base la instruccion mul $t7, $t6, $t0, intentaba leer el valor del registro $t6 justo despues de la instruccion lw $t6, 0($t5), en la arquitectura segmentada, el dato cargado desde memoria NO esta disponible sino hasta despues de la etapa MEM o inicios de WB, esto genera el riesgo (Load-Use), generando asi una parada de un ciclo.

> Para optimizar se movio la intruccion que incrementa el indice (addi $t3, $t3, 1) para que se ejecuto justo entre el lw y el mul, esto permite que el procesador realice un trabajo util mientras que la memoria esta procesando la carga del dato, eliminando asi la necesidad de la parada, reduciendo los ciclos por iteración y evitando los tiempos muertos del procesador

---

3. Comparativa de Resultados

Métrica	Código Base	Código Optimizado	Mejora (%)
Ciclos Totales	110	102	7.2%
Stalls (Paradas)	16	8	50%
CPI	1.17	1.08	7.7%

---

4. Conclusiones

¿Qué impacto tiene la segmentación en el diseño de software de bajo nivel? ¿Es siempre posible eliminar todas las paradas?

> **¿Qué impacto tiene la segmentación en el diseño de software de bajo nivel?**: Esto obliga al programador a seer consciente del orden de las instrucciones, ya que afecta directamente el rendimiento del hardware y no solamente la lógica, con esto se puede concluir que un código que ignora las dependencias de datos, resultara en un CPI elevado afectameente directamente el potncia de la CPU debido a las constantes paradas de hardware

> **¿Es siempre posible eliminar todas las paradas?**: No, no siempre se podra eliminar todas las paradas, debido que existiran escenarios donde algunas instrucciones dependeran de resultados inmediatos