Asignatura	Datos del alumno	Fecha
Estructura de Computadores	Apellidos: Bohorquez Calderon	17/03/25
	Nombre: Santiago	

Laboratorio #1: Simulación y optimización de un programa en un procesador escalar segmentado

Introducción

Este informe documenta la implementación de algoritmos en ensamblador MIPS con la herramienta MARS. El propósito fue desarrollar y ejecutar scripts en ensamblador para resolver problemas concretos, explorando el funcionamiento de bajo nivel de un procesador y la manipulación de registros y memoria.

Se crearon tres scripts:

*Un script para identificar el número mayor en una lista de valores proporcionada por el usuario.

*Un script para identificar el número menor en la misma lista de valores.

*Un script para generar la serie Fibonacci hasta un límite definido por el usuario, y calcular la suma de los números generados.

Desarrollo de la actividad

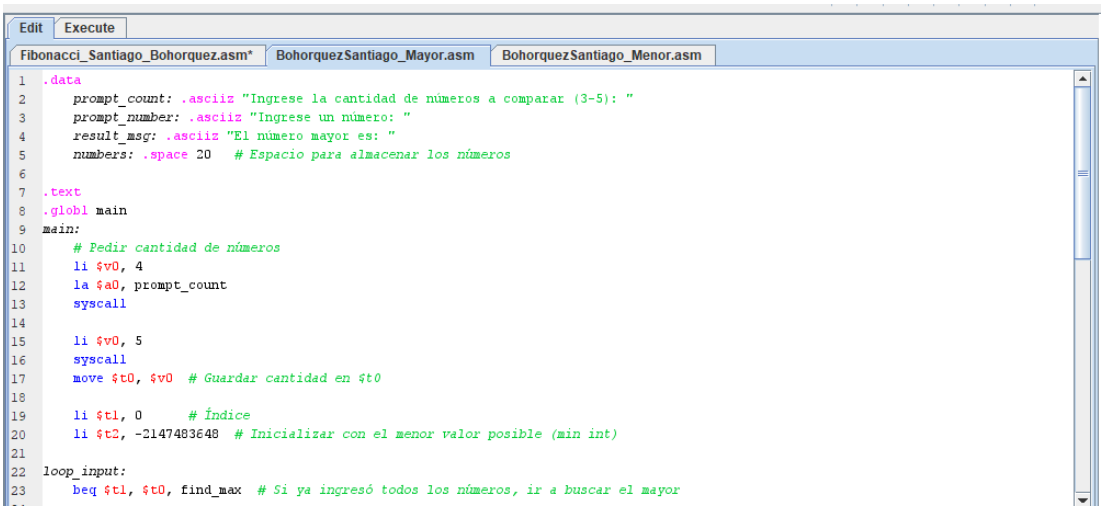
Para la ejecución de esta actividad, se utilizó el simulador MARS (MIPS Assembler and Runtime Simulator), el cual permite compilar y ejecutar programas escritos en lenguaje ensamblador MIPS. A continuación, se detallan los pasos seguidos en la implementación de cada uno de los scripts:

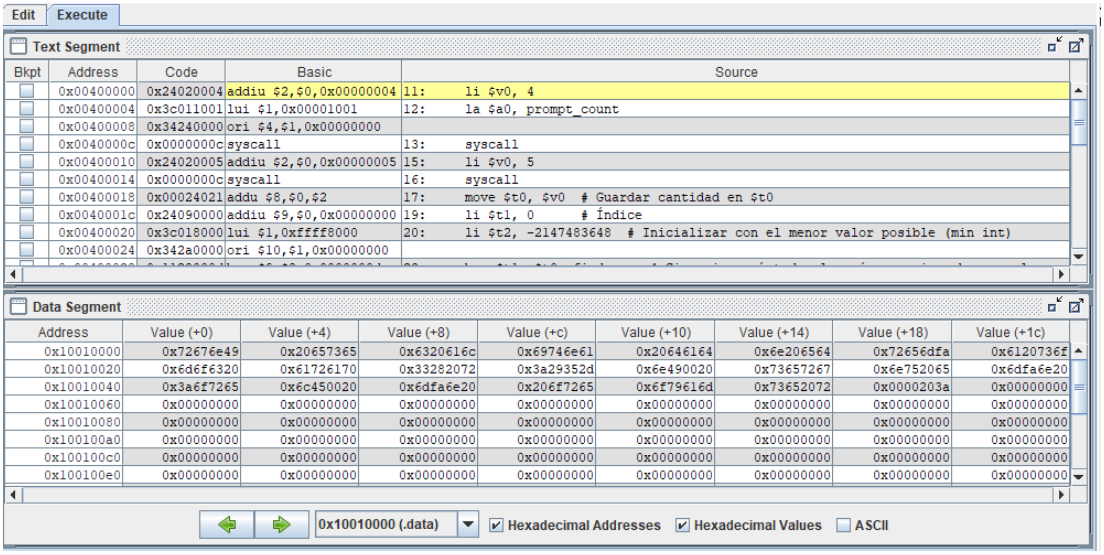
1. Identificación del número mayor

- Se solicitó al usuario ingresar una cantidad de números entre 3 y 5.
- Se almacenaron los valores en registros.
- Se compararon los valores mediante instrucciones de comparación y salto condicional.
- Se imprimió el número mayor en la consola.

Asignatura	Datos del alumno	Fecha
Estructura de Computadores	Apellidos: Bohorquez Calderon	17/03/25
	Nombre: Santiago	

Capturas de pantalla





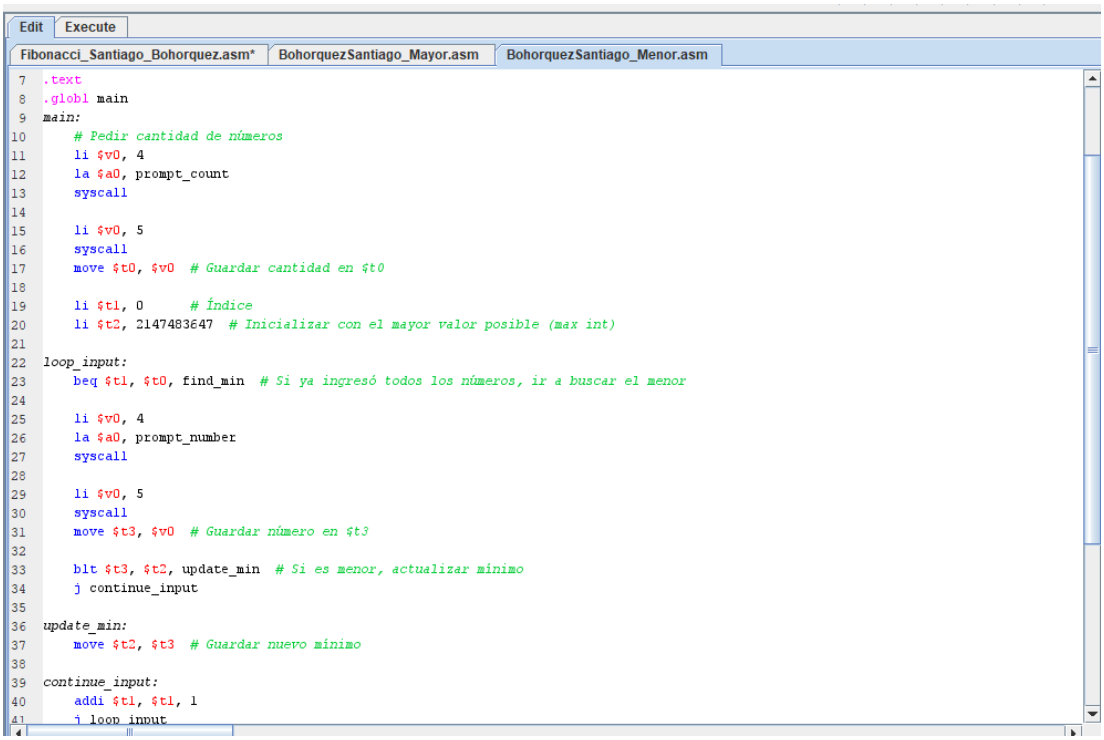
Asignatura	Datos del alumno	Fecha
Estructura de Computadores	Apellidos: Bohorquez Calderon	17/03/25
	Nombre: Santiago	

```
Ingrese la cantidad de números a comparar (3-5): 5
Ingrese un número: 4
Ingrese un número: 5
Ingrese un número: 9
Ingrese un número: 15
Ingrese un número: 16
El número mayor es: 16
-- program is finished running --
```

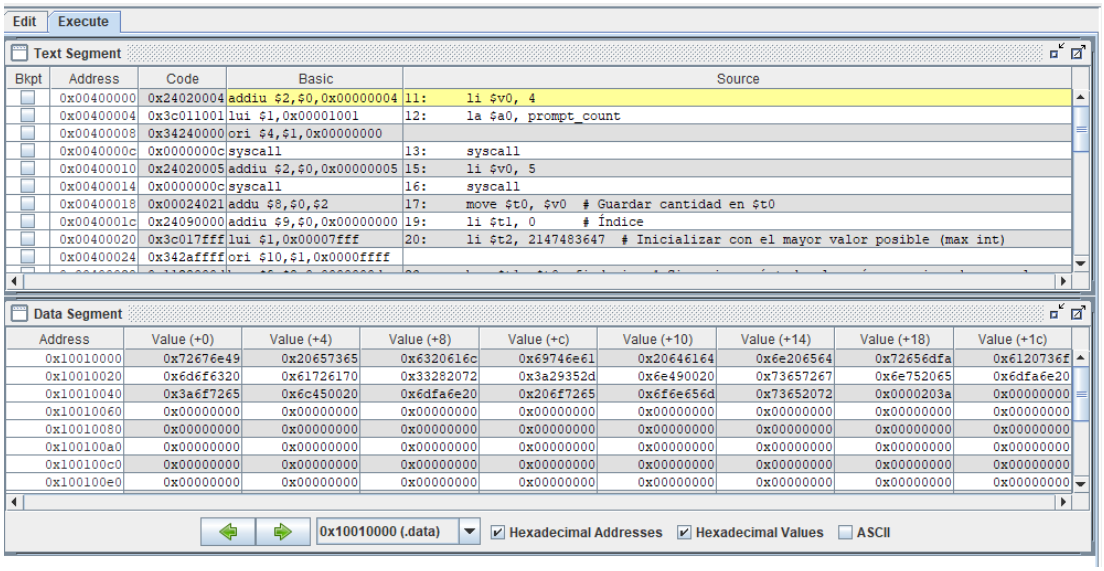
2. Identificación del número menor

- Se solicitó al usuario ingresar una cantidad de números entre 3 y 5.
- Se compararon los valores almacenados en registros.
- Se imprimió el número menor en la consola.

Capturas de pantalla



Asignatura	Datos del alumno	Fecha
Estructura de Computadores	Apellidos: Bohorquez Calderon	17/03/25
	Nombre: Santiago	




```
Ingrese la cantidad de números a comparar (3-5): 3
Ingrese un número: 5
Ingrese un número: 15
Ingrese un número: 16
El número menor es: 5
-- program is finished running --
```

3. Generación de la serie Fibonacci

- Se pidió al usuario ingresar la cantidad de términos a generar, con un límite máximo de 12.
- Se calculó la serie Fibonacci usando sumas sucesivas y almacenamiento en registros.
- Se imprimieron los valores de la serie junto con su suma total.

Asignatura	Datos del alumno	Fecha
Estructura de Computadores	Apellidos: Bohorquez Calderon	17/03/25
	Nombre: Santiago	

Capturas de pantalla

Fibonacci_Santiago_Bohorquez.asm* | BohorquezSantiago_Mayor.asm | BohorquezSantiago_Menor.asm

```
.data
    prompt: .asciiz "Ingrese la cantidad de números de la serie Fibonacci"
    error_msg: .asciiz "Número no válido. Intente nuevamente.\n"
    result_msg: .asciiz "Serie Fibonacci: "
    sum_msg: .asciiz "\nSuma total: "

.text
.globl main
main:
    # Pedir la cantidad de números
ask_again:
    li $v0, 4
    la $a0, prompt
    syscall

    li $v0, 5
    syscall
    move $t0, $v0  # Guardar cantidad en $t0

    # Validar que esté entre 1 y 12
    blt $t0, 1, invalid_input
    bgt $t0, 12, invalid_input
    j continue_execution

invalid_input:
    li $v0, 4
    la $a0, error_msg
    syscall
    j ask_again  # Volver a pedir el número

continue_execution:
    # Mostrar mensaje de inicio
    li $v0, 4
    la $a0, result_msg
    syscall
```

Edit | Execute

Text Segment

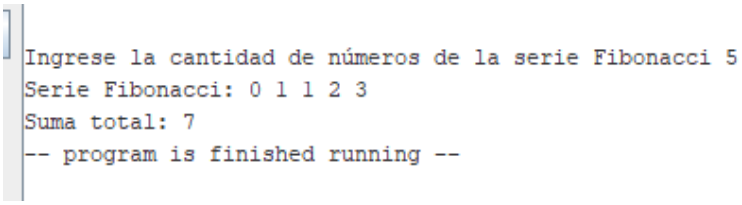
Bkpt	Address	Code	Basic	Source
	0x00400000	0x24020004	addiu $2,$0,0x00000004	12: li $v0, 4
	0x00400004	0x3c011001	lui $1,0x00001001	13: la $a0, prompt
	0x00400008	0x34240000	ori $4,$1,0x00000000	
	0x0040000c	0x0000000c	syscall	14: syscall
	0x00400010	0x24020005	addiu $2,$0,0x00000005	16: li $v0, 5
	0x00400014	0x0000000c	syscall	17: syscall
	0x00400018	0x00024021	addu $8,$0,$2	18: move $t0, $v0 # Guardar cantidad en $t0
	0x0040001c	0x29010001	slti $1,$8,0x00000001	21: blt $t0, 1, invalid_input
	0x00400020	0x14200004	bne $1,$0,0x00000004	
	0x00400024	0x2001000c	addi $1,$0,0x0000000c	22: bgt $t0, 12, invalid_input

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x72676e49	0x20657365	0x6320616c	0x69746e61	0x20646164	0x6e206564	0x72656dfa	0x6420736f
0x10010020	0x616c2065	0x72657320	0x46206569	0x6e6f6269	0x69636361	0xfa4e0020	0x6f72656d	0x206f6e20
0x10010040	0x696ce176	0x202e6f64	0x65746e49	0x2065746e	0x7665756e	0x6e656d61	0x0a2e6574	0x72655300
0x10010060	0x46206569	0x6e6f6269	0x69636361	0x0a00203a	0x616d7553	0x746f7420	0x203a6c61	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) ☑ Hexadecimal Addresses ☑ Hexadecimal Values ☐ ASCII

Asignatura	Datos del alumno	Fecha
Estructura de Computadores	Apellidos: Bohorquez Calderon	17/03/25
	Nombre: Santiago	



Conclusiones

La implementación de algoritmos en ensamblador fue un ejercicio valioso para entender la arquitectura MIPS y sus instrucciones básicas. Pude apreciar la diferencia entre la programación de bajo y alto nivel, y la importancia de optimizar el código