Practica 1

Torres de Hanói

-Juan Pablo Figueroa Martín 750901.

-Luis Pablo de los Reyes Rentería 750515.

-Carrera: Ingeniería en Ciberseguridad.

- Maestro: Álvaro Gutiérrez Arce.

-Curso: Organización y Arquitectura de Computadoras.

-Fecha: 27 de Octubre del 2024.

ITESO, Universidad Jesuita de Guadalajara

Índice

Diagrama de flujo del programa implementado realizado en Visio o programa equivalente.

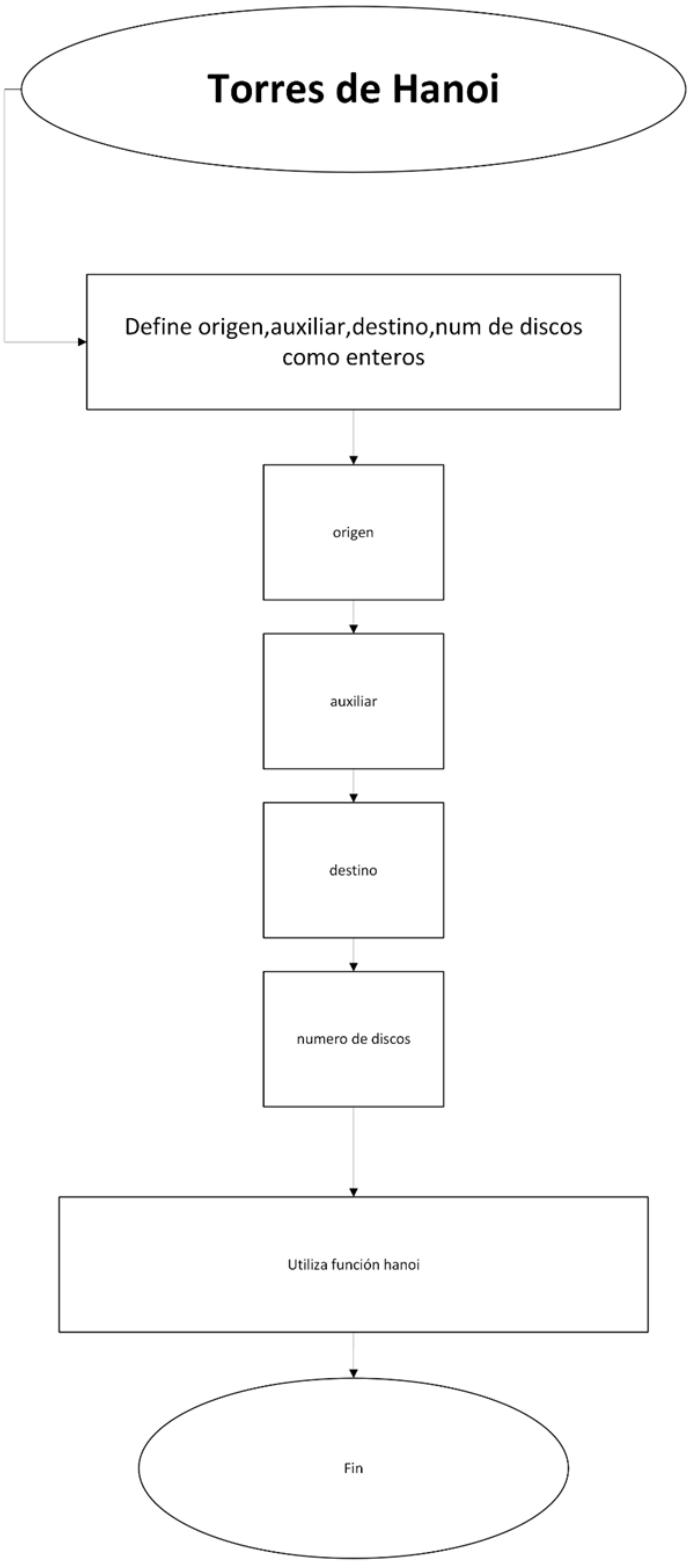


Figura 1. Diagrama de Flujo 1.

Las decisiones que se tomaron al diseñar su programa, se puede tomar como referencia el diagrama de flujo del programa.

Diseñamos el programa en base al algoritmo presentado en el diagrama de flujo, apoyándonos en él para poder estructurar el código de una manera más sencilla y eficiente.

Nos enfocamos en hacer que nuestro código funcionara mediante recursividad para así poder hacer que se utilice la menor memoria posible al momento de estar corriendo el algoritmo.

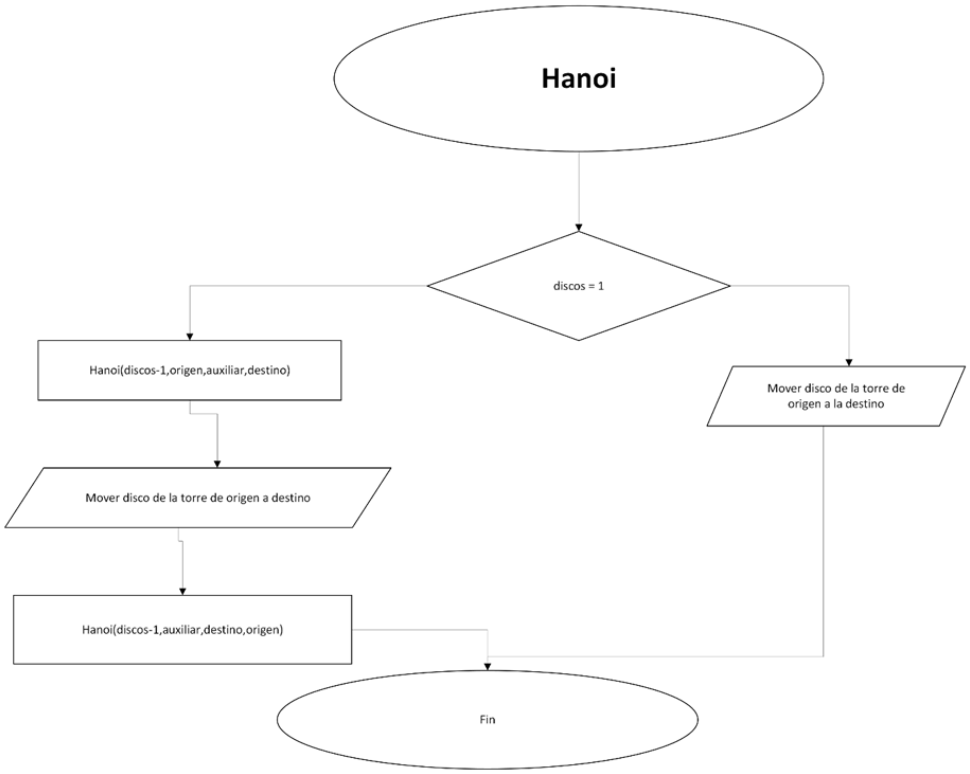
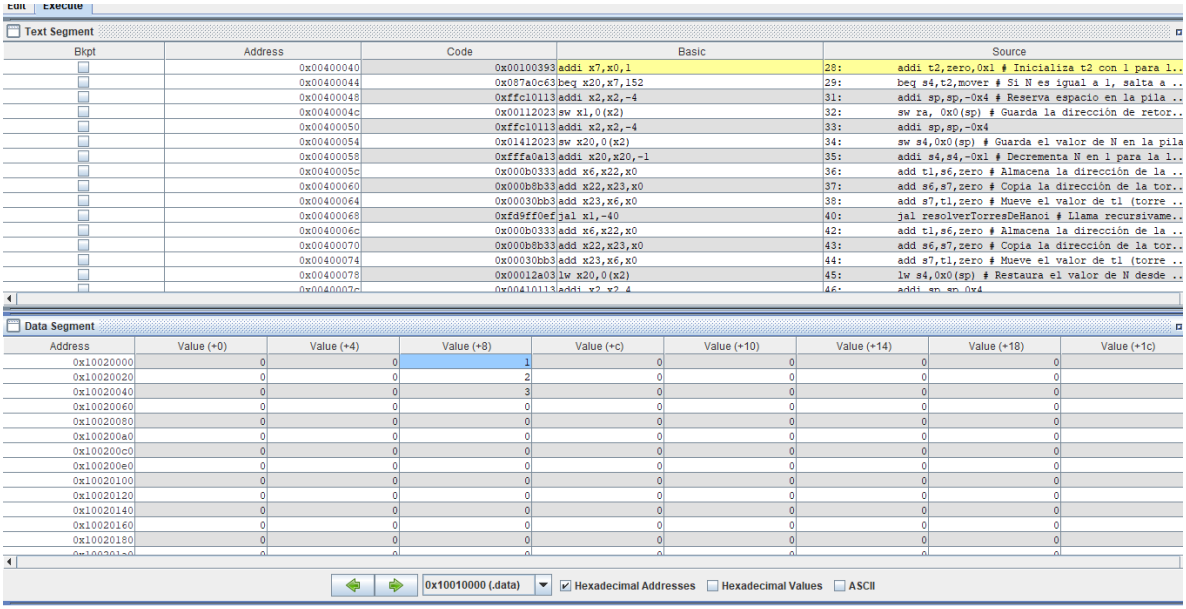


Figura 2. Diagrama de Flujo 2.

Una simulación en el RARS para 3 discos (La simulación son impresiones de pantalla de data segment del RARS) .



Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400040	0x00100393	addi x7,x0,1	28: addi t2,zero,0x1 # Inicializa t2 con 1 para l..
	0x00400044	0x087a0c63	beq x20,x7,152	29: beq s4,t2,mover # Si N es igual a 1, salta a ..
	0x00400048	0xffc10113	addi x2,x2,-4	31: addi sp,sp,-0x4 # Reserva espacio en la pila ..
	0x0040004c	0x00112023	sw x1,0(x2)	32: sw ra, 0x0(sp) # Guarda la dirección de retor..
	0x00400050	0xffc10113	addi x2,x2,-4	33: addi sp,sp,-0x4
	0x00400054	0x01412023	sw x20,0(x2)	34: sw s4,0x0(sp) # Guarda el valor de N en la pila
	0x00400058	0xfffa0a13	addi x20,x20,-1	35: addi s4,s4,-0x1 # Decrementa N en 1 para la 1..
	0x0040005c	0x000b0333	add x6,x22,x0	36: add t1,s6,zero # Almacena la dirección de la ..
	0x00400060	0x000b8b33	add x22,x23,x0	37: add s6,s7,zero # Copia la dirección de la tor..
	0x00400064	0x00030bb3	add x23,x6,x0	38: add s7,t1,zero # Mueve el valor de t1 (torre ..
	0x00400068	0xfd9ff0ef	jal x1,-40	40: jal resolverTorresDeHanoi # Llama recursivame..
	0x0040006c	0x000b0333	add x6,x22,x0	42: add t1,s6,zero # Almacena la dirección de la ..
	0x00400070	0x000b8b33	add x22,x23,x0	43: add s6,s7,zero # Copia la dirección de la tor..
	0x00400074	0x00030bb3	add x23,x6,x0	44: add s7,t1,zero # Mueve el valor de t1 (torre ..
	0x00400078	0x00012a03	lw x20,0(x2)	45: lw s4,0x0(sp) # Restaura el valor de N desde ..
	0x0040007c	0x00410113	addi x2,x2,4	46: addi sp,sp,0x4

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10020000	0	0	1	0	0	0	0	
0x10020020	0	0	2	0	0	0	0	
0x10020040	0	0	3	0	0	0	0	
0x10020060	0	0	0	0	0	0	0	
0x10020080	0	0	0	0	0	0	0	
0x100200a0	0	0	0	0	0	0	0	
0x100200c0	0	0	0	0	0	0	0	
0x100200e0	0	0	0	0	0	0	0	
0x10020100	0	0	0	0	0	0	0	
0x10020120	0	0	0	0	0	0	0	
0x10020140	0	0	0	0	0	0	0	
0x10020160	0	0	0	0	0	0	0	
0x10020180	0	0	0	0	0	0	0	

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Figura 3. Simulación de 3 Discos en RARS.

Análisis del comportamiento del stack para el caso de 3 discos.

Con 3 discos, el programa hace llamadas recursivas, donde cada nivel de recursión guarda en el stack los valores de los registros ra, s0, y a0. Esto es necesario para preservar el contexto en cada llamada a hanoi, y luego restaurarlo cuando regresa de la recursión.

- **Primer movimiento (n = 3)**:
    - Al entrar en hanoi, se decrementa el número de discos (s0 = 3 -> 2), y la función llama recursivamente a hanoi para mover 2 discos de torre_origen a torre_auxiliar.
    - Esto guarda el contexto en el stack.
- **Segundo movimiento (n = 2)**:
    - Ahora, hanoi es llamado con n = 2, por lo que el contexto se guarda nuevamente en el stack.
    - La recursión continúa hasta que n = 1.
- **Caso base (n = 1)**:
    - Al llegar a n = 1, se ejecuta la instrucción mover_disco, y el disco se mueve directamente.
    - Tras mover el disco, el programa regresa a las llamadas anteriores, restaurando el contexto en cada retorno del stack.

Para 3 discos, el stack se llena y se vacía en cada llamada y retorno, con una profundidad máxima igual a la cantidad de discos, más el número de veces que la función mover_disco se llama de forma recursiva.

Incluir en el instruction count (IC) y especificar el porcentaje de instrucciones de tipo R, I y J para 8 discos.

Instruction Count (IC):

- Con 8 discos, la complejidad de las Torres de Hanói crece exponencialmente. Para cada movimiento del disco, el código ejecuta las instrucciones de stack, movimiento, y las instrucciones condicionales de salto y comparación.

Distribución de instrucciones:

1. **Tipo I**: Instrucciones de carga y almacenamiento (lw, sw), addi, y comparaciones (beq). Estas representan la mayor parte de las instrucciones, pues la carga y el almacenamiento en el stack ocurren en cada llamada recursiva.
2. **Tipo J**: Instrucciones de salto, principalmente jal para las llamadas recursivas y j para el bucle infinito. Estas son pocas en comparación a los otros tipos.
3. **Tipo R**: Operaciones aritméticas o de manipulación de datos (si las hubiera, aunque en este código son mínimas).

Una gráfica que muestre como se incrementa el IC para las torres de Hanói en los casos de 4 a 15 discos.

La grafica que se mostrara para resolver el problema de las torres de Hanoi, será una gráfica exponencial, debido a que el número mínimo de movimientos necesarios para resolverlo es $2^n - 1$, tomando "n" como el número de discos que queremos, en este caso de 4 a 15.

Esto lo podemos ver en la siguiente grafica elaborada en Excel.

Luis Pablo De los Reyes Rentería 750515
Juan Pablo Figueroa Martín 750901

Gráfica que muestre como se incrementa el IC para las torres de hanoi en los casos de 4 a 15 discos.

Numero de Discos	Instruction Count	$= IC = 2^n - 1$
4	15	
5	31	
6	63	
7	127	
8	255	
9	511	
10	1023	
11	2047	
12	4095	
13	8191	
14	16383	
15	32767	

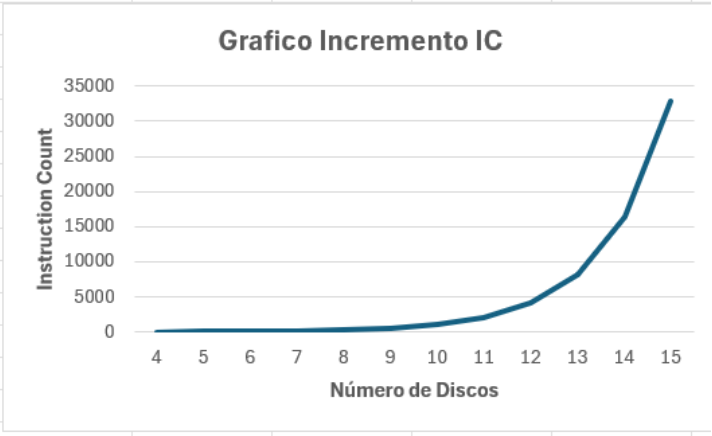


Figura 4. Gráfico del Incremento del IC para las Torres de Hanoi.

Conclusiones de cada integrante del equipo.

Luis Pablo De los Reyes Rentería:

En esta práctica de Torres de Hanói, uno de los mayores retos fue manejar la recursión y el stack correctamente. Al inicio, tuvimos problemas con el orden de los registros, lo que hacía que el programa no funcionara como esperábamos. Sin embargo, con ajustes y pruebas, logramos estabilizar el código y documentarlo de manera clara. Fue una buena experiencia para entender mejor cómo funciona el stack en ensamblador. También quiero dar gracias al profesor por como imparte sus clases, siempre favoreciendo el solucionar las dudas y siento que fue algo muy útil para hacer el proyecto.

Juan Pablo Figueroa Martín:

Con esta práctica nos ayudó a trabajar en detalle con ensamblador y el uso eficiente del stack. Aunque parecía sencillo, ajustar correctamente las llamadas recursivas y el manejo de los registros resultó complicado; cada ajuste en el orden de registro requería pruebas cuidadosas. A pesar de que al principio nos equivocamos demasiado, logramos que funcionara, también la documentación del código fue algo que antes no tomábamos tanto cuidado en lenguajes de medio o alto nivel, pero ahora lo tuvimos que hacer porque al principio nos revolvíamos mucho en que era lo que hacía cada línea de nuestro código.

Link de GitHub:
https://github.com/750901-750515