

Práctica 1 – Organización y Arquitectura de Computadoras

Torres de Hanoi



ITESO, Universidad
Jesuita de Guadalajara

Diego Arturo Orozco Sánchez – 739246

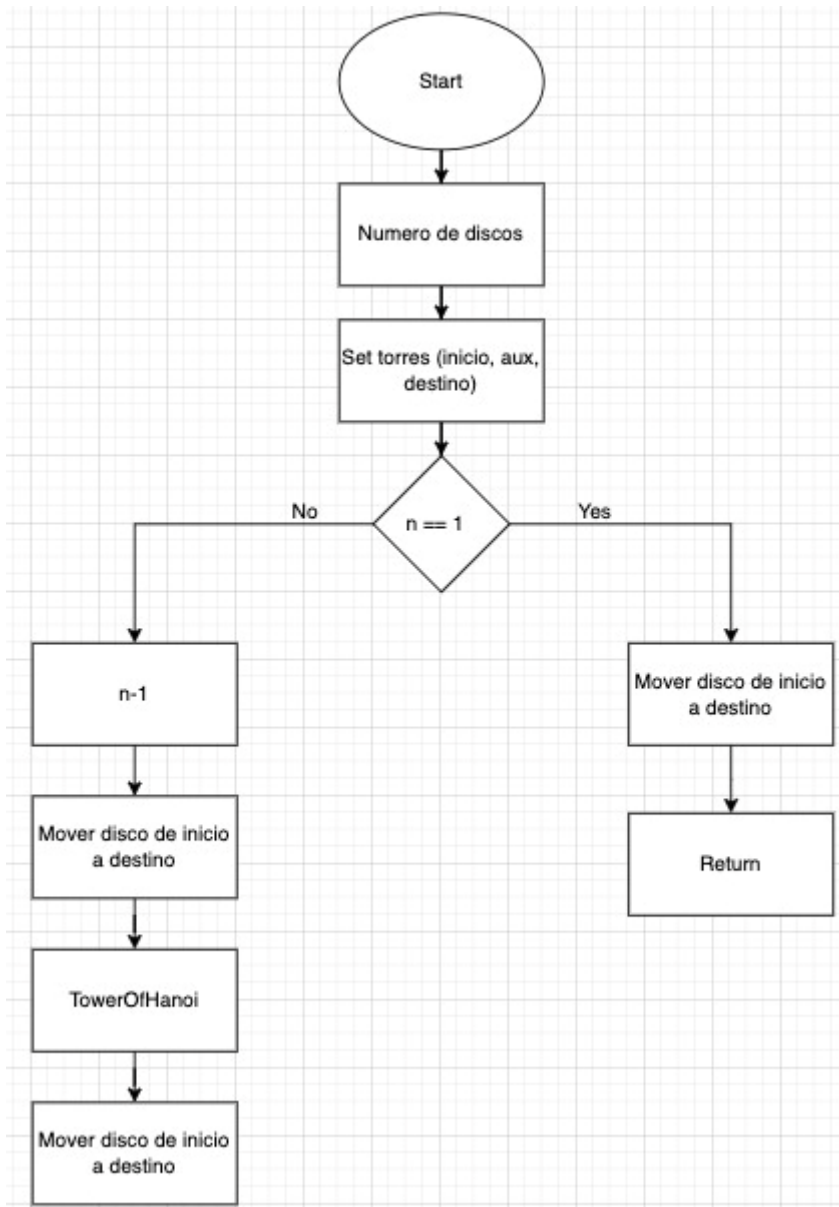
Isaac Ramírez Robles-Castillo – 748062

Profesor:

Juanpablo Ibarra Esparza

Fecha: 31/10/2023

Diagrama de flujo:



Decisiones de diseño:

Principalmente nos basamos en el código de Fibonacci que hicimos en clase, además del algoritmo de C que tomamos como referencia.

En el main, primero definimos todas las variables que vamos a usar en el programa, como la inicialización de la RAM usando lui, el número de discos, la variable del ciclo for al llenar la primera torre, y demás.

Primero se llena la primera torre usando una simulación de un ciclo for, que se llama a sí mismo recursivamente hasta que termine.

Después buscamos el caso default del algoritmo (dónde solo queda un disco), y el else nos lleva a la recursión fuerte del algoritmo. A continuación, se especifica cómo funciona la recursión y el stack.

Simulación con 3 discos:

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400034	0x00448493	addi x9,x9,4	21: addi s1, s1, 4 ...
	0x00400038	0x00490913	addi x18,x18,4	22: addi s2, s2, 4
	0x0040003c	0x00498993	addi x19,x19,4	23: addi s3, s3, 4
	0x00400040	0x001a0a13	addi x20,x20,1	24: addi s4, s4, 1 ...
	0x00400044	0xfe9ff0ef	jal x1,-24	25: jal for
	0x00400048	0xffc48493	addi x9,x9,-4	28: addi s1, s1, -4 ...
	0x0040004c	0x008000ef	jal x1,8	30: jal hanoi ...
	0x00400050	0x0d4000ef	jal x1,212	32: jal exit ...
	0x00400054	0x02639063	bne x7,x6,32	36: bne t2, t1, else # Cuand...

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	1	2	3	0	0	0	0	0
0x10010020	0	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400104	0xf51ff0ef	jal x1,-176	105: jal hanoi # Segun...
	0x00400108	0x00012383	lw x7,0(x2)	108: lw t2, 0(sp)
	0x0040010c	0x00412083	lw x1,4(x2)	109: lw ra, 4(sp)
	0x00400110	0x00812483	lw x9,8(x2)	110: lw s1, 8(sp)
	0x00400114	0x00c12903	lw x18,12(x2)	111: lw s2, 12(sp)
	0x00400118	0x01012983	lw x19,16(x2)	112: lw s3, 16(sp)
	0x0040011c	0x01410113	addi x2,x2,20	113: addi sp, sp, 20
	0x00400120	0x000080e7	jalr x1,x1,0	115: jalr ra # Retor...
	0x00400124	0x00000013	addi x0,x0,0	117: exit: nop

Data Segment

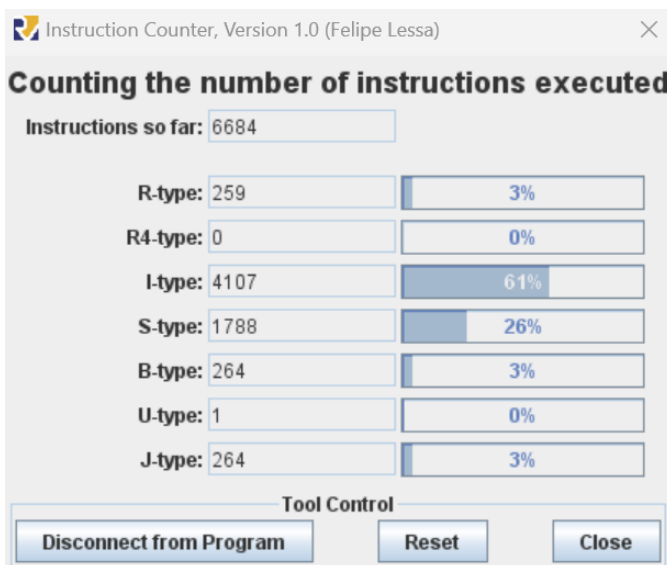
Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0	0	0	0	0	0	1	2
0x10010020	3	0	0	0	0	0	0	0
0x10010040	0	0	0	0	0	0	0	0
0x10010060	0	0	0	0	0	0	0	0
0x10010080	0	0	0	0	0	0	0	0
0x100100a0	0	0	0	0	0	0	0	0
0x100100c0	0	0	0	0	0	0	0	0

0x10010000 (.data) Hexadecimal Addresses Hexadecimal Values ASCII

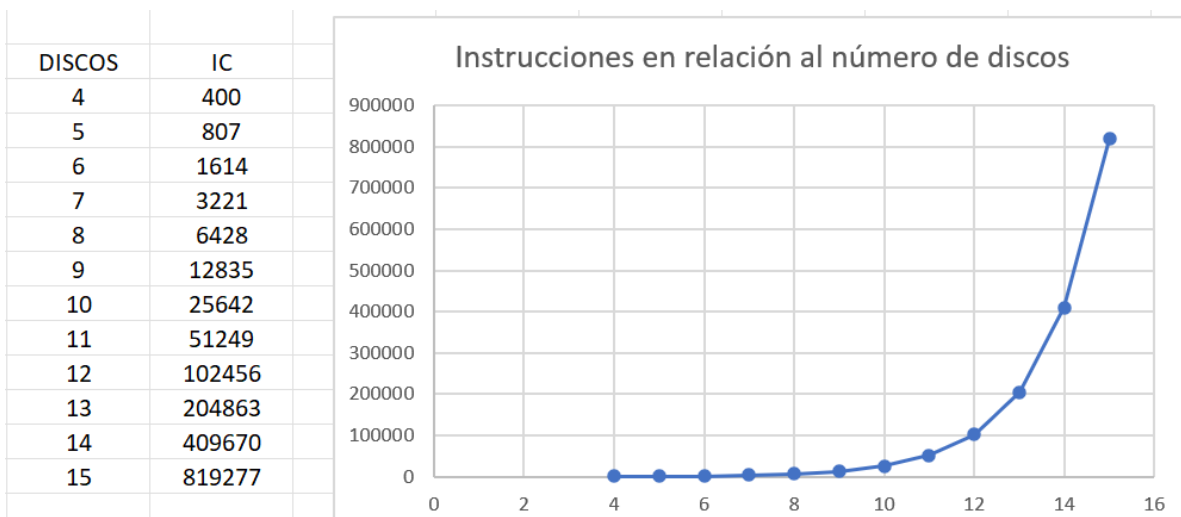
Análisis del comportamiento del stack con 3 discos:

Para entender lo que hace el stack con 3 discos, debemos referirnos a la parte del código dónde empezamos la recursión. Se reservaron 20 bits en el stack, pues necesitamos 4 bits para 5 registros, y lo primero de que debemos hacer es el push con sw como tal. Después, con esa memoria liberada, se hacen los movimientos necesarios para al final reponer la memoria con lw.

Instruction count para 8 discos:



Gráfica del instruction count de 4 a 15 discos:



Conclusiones:

Diego: Personalmente, creo que haber aprendido ensamblador es bueno para saberlo, pero como ingenieros en sistemas creo que está de más. La programación en ensamblador es bastante diferente a lo que yo estoy acostumbrado, y honestamente nos costó bastante. Como tal, el algoritmo de C en el que nos basamos es fácil de entender, y también el problema de las Torres de Hanoi, pero traducirlo a ensamblador y si fue una práctica difícil. El saber como manejar los registros, las variables temporales, manejo de memoria del stack, y demás conocimientos que eran muy importantes para saber implementarlo. Sin embargo, me gustó probar mis habilidades en un reto así de demandante.

Isaac: En el desarrollo de la práctica se tuvo analizar a detalle el algoritmo de Torres de Hanoi para plasmarlo a un esquema de ensamblador, se profundizó en el uso de variables (declaraciones y guardado de información), manejos de memoria para guardar las posiciones de cada variable y poder interpretar los resultados "gráficamente". Se uso recursividad para poder implementar un código eficiente, practico y poder usar el número de discos que se necesiten. Personalmente encontré muy desafiante la actividad debido a la diferencia de lógica que se necesita para pasar un código a ensamblador y el uso de la memoria me pareció complicado ya que se necesita tener una buena noción de los comportamientos en el stack.