



# ITESO

Universidad Jesuita  
de Guadalajara

**“Proyecto Torres de Hanoi”**

Materia:

Organización y Arquitectura de  
computadoras.

Participantes  
(con  
expediente):

Renata Itzel Caballero Velázquez.  
741036  
Getsemani Sarahi Monreal Tapia  
744037

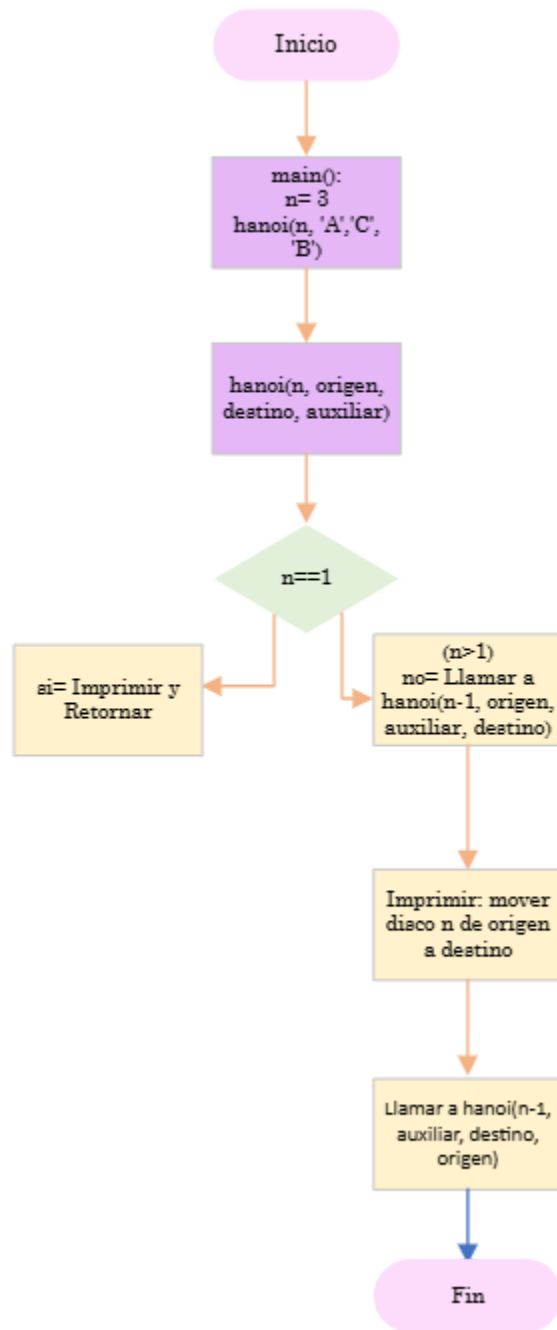
Profesor:

JuanPablo Ibarra Esparza

Fecha:

Jueves 20 de junio 2024

## Diagrama de Flujo



## Decisiones en la Implementación del Código de las Torres de Hanoi

Decidimos utilizar una función recursiva para resolver el problema ya que el problema de las Torres de Hanoi se define de manera natural de forma recursiva (aparte que era obligatorio), esta simplifica la solución, dividiendo el problema en subproblemas más pequeños del mismo tipo.

Existía la alternativa de una implementación iterativa. Sin embargo, para esta ocasión tenía que ser de forma recursiva.

Estructura de la Función hanoi:

Definimos una función `hanoi` que toma cuatro parámetros: el número de discos `n`, la torre de origen `origen`, la torre de destino `destino` y la torre auxiliar `auxiliar`.

Estos parámetros son esenciales para representar el estado actual del problema. Permiten identificar desde dónde se mueve el disco y hacia dónde, y cuál es la torre auxiliar utilizada para facilitar el movimiento. Podríamos haber usado variables globales, pero eso haría el código más difícil de seguir y depurar.

Condición Base de la Recursión:

Implementamos una condición base `if (n == 1)`.

La condición base es necesaria para evitar una recursión infinita. En el caso de `n == 1`, el problema es trivial (mover un solo disco) y no requiere más divisiones.

Si no se implementara una condición base adecuada, la función recursiva entraría en un bucle infinito, resultando en un desbordamiento de la pila.

Llamadas Recursivas:

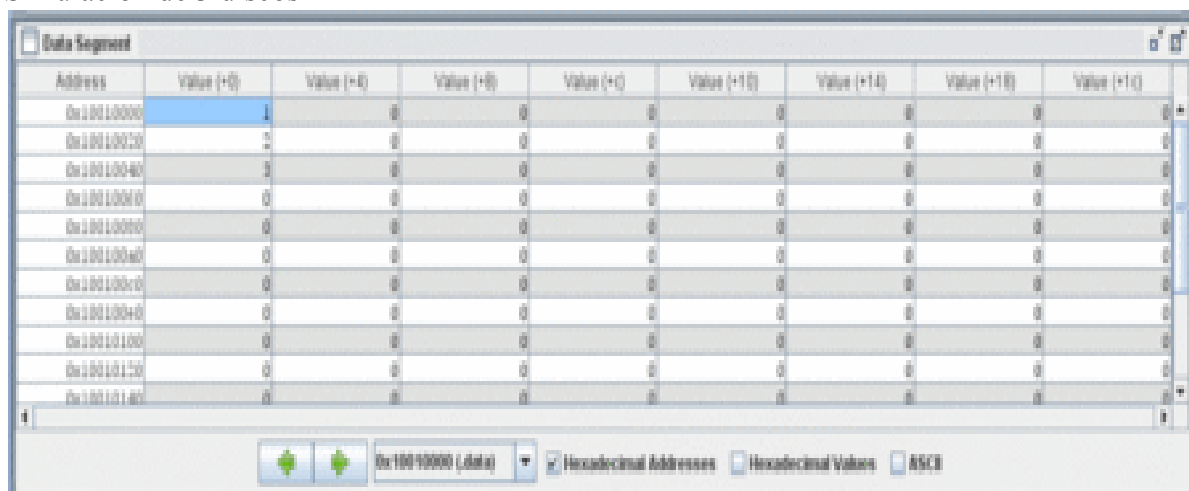
Realizamos dos llamadas recursivas dentro de la función `hanoi` para mover `n-1` discos.

1. La primera llamada mueve `n-1` discos de la torre de origen a la torre auxiliar.
2. Luego se mueve el disco `n` de la torre de origen a la torre de destino.
3. La segunda llamada mueve los `n-1` discos de la torre auxiliar a la torre de destino.

Se podría usar un algoritmo no recursivo, pero esto complicaría la lógica y la implementación.

Al final se encuentra la impresión de los movimientos.

## Simulación de 3 discos



Address	Value (+0)	Value (+4)	Value (+8)	Value (+C)	Value (+10)	Value (+14)	Value (+18)	Value (+1C)
0x00100000	1	0	0	0	0	0	0	0
0x00100020	2	0	0	0	0	0	0	0
0x00100040	3	0	0	0	0	0	0	0
0x00100060	0	0	0	0	0	0	0	0
0x00100080	0	0	0	0	0	0	0	0
0x001000A0	0	0	0	0	0	0	0	0
0x001000C0	0	0	0	0	0	0	0	0
0x001000E0	0	0	0	0	0	0	0	0
0x00100100	0	0	0	0	0	0	0	0
0x00100120	0	0	0	0	0	0	0	0
0x00100140	0	0	0	0	0	0	0	0

Análisis del comportamiento del stack para 3 discos:

Para el caso de 3 discos, el comportamiento del stack es el siguiente:

1. Inicialización:
  - El stack pointer (sp) comienza en su posición inicial.
2. Primera llamada a `hanoi`:
  - Se hace push de ra (dirección de retorno) al stack.
  - sp se decrementa en 4 bytes.
3. Segunda llamada a `hanoi` (2 discos):
  - Se hace push de ra al stack.
  - sp se decrementa en 4 bytes más.
4. Tercera llamada a `hanoi` (1 disco):
  - Se hace push de ra al stack.
  - sp se decrementa en 4 bytes más.
5. Se alcanza el caso base (1 disco):
  - No se realiza push al stack.
  - Se ejecuta `swap`.
6. Retorno de la tercera llamada:
  - Se hace pop de ra del stack.
  - sp se incrementa en 4 bytes.
7. Segunda llamada a `hanoi` dentro de la segunda llamada:
  - Se hace push de ra al stack.
  - sp se decrementa en 4 bytes.
8. Se alcanza el caso base nuevamente:
  - No se realiza push al stack.
  - Se ejecuta `swap`.
9. Retorno de la segunda llamada interna:
  - Se hace pop de ra del stack.
  - sp se incrementa en 4 bytes.
10. Retorno de la segunda llamada original:
  - Se hace pop de ra del stack.
  - sp se incrementa en 4 bytes.
11. Segunda llamada a `hanoi` dentro de la primera llamada:
  - Se hace push de ra al stack.
  - sp se decrementa en 4 bytes.

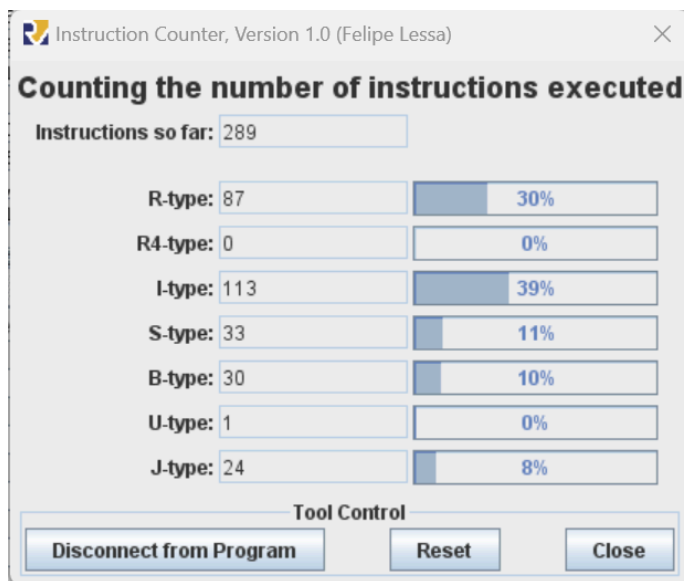
12. El proceso se repite de manera similar para esta llamada.

13. Finalmente, retorno de la primera llamada:

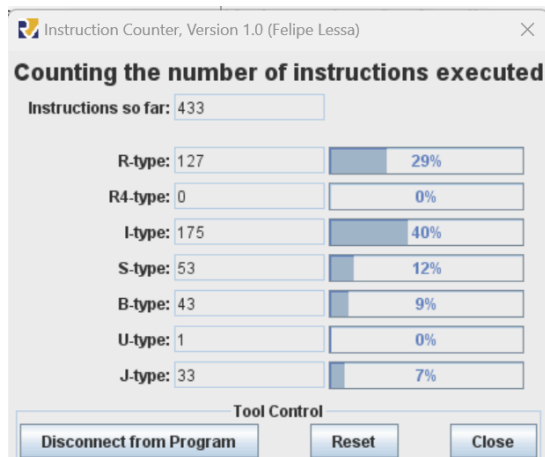
- Se hace pop del último ra del stack.
- sp vuelve a su posición original.

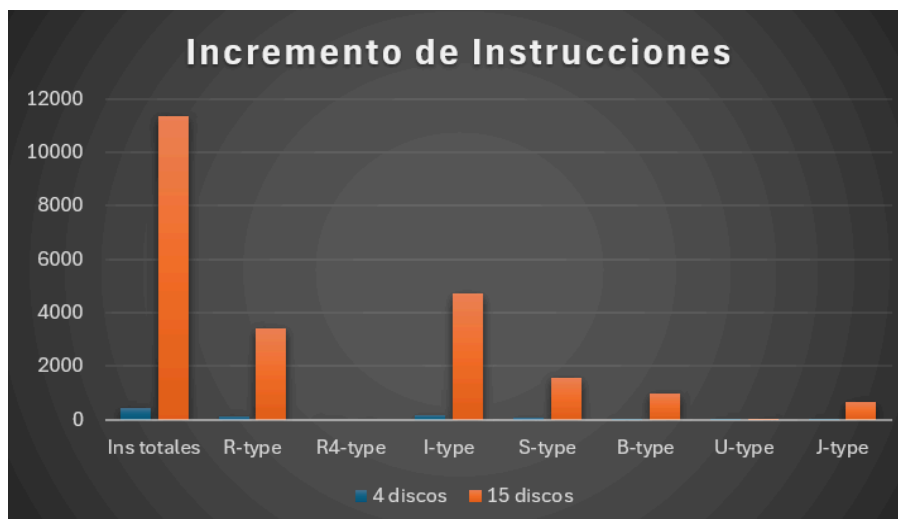
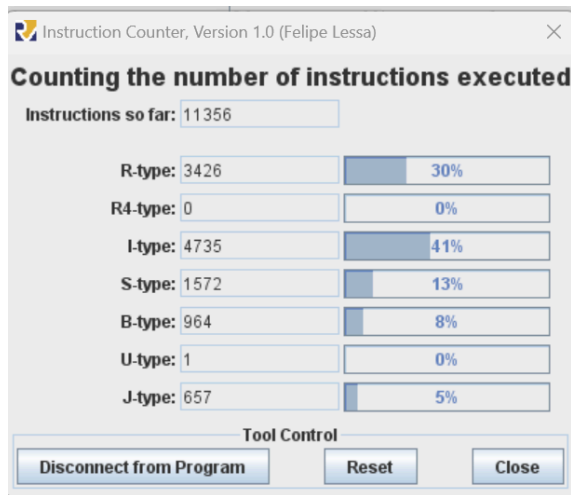
En resumen, para 3 discos, el stack alcanzará una profundidad máxima de 3 niveles (12 bytes), correspondientes a las llamadas recursivas anidadas de la función `hanoi`; el stack crecerá y se contraerá a medida que se realizan las llamadas recursivas y los retornos. Este patrón de uso del stack garantiza que cada llamada a la función tenga su propia copia de la dirección de retorno, permitiendo que la recursión funcione correctamente y que el programa pueda volver al punto correcto después de cada llamada.

## Instruction Count



## Instruction Count de 4 a 15 discos





## Conclusiones:

Renata.

En lo personal fue algo compleja y tediosa la realización de esta práctica, más que nada por el hecho que es un lenguaje nuevo y diferente aparte de que es bastante información en poco tiempo; el estar haciendo ejercicios y revisar los ejemplos de clase nos ayudo bastante para guiarnos en el proceso.

Getsemani.

Para mi esta practica es de las mas complicadas que he hecho, realmente me hizo tenerle mucho respeto al lenguaje de ensamblador, nos guiamos mucho del ejemplo de fibonacci que hicimos en clase pero nuestro mayor reto fue hacer las modificaciones necesarias para que el codigo funcionara en base al codigo inicial de C y mantener la recursividad, fue un gran reto pero a pesar de los muchos codigos fallidos y la complejidad de la practica al final la pudimos terminar de forma funcional, fue un proyecto dificil pero valio la pena y disfrute el aprender un nuevo lenguaje.