PRACTICA 1.Torres de Hanói

ITESO, Universidad Jesuita de Guadalajara

ITESO, Universidad Jesuita de Guadalajara

MATERIA: ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

PROFESOR: JUANPABLO IBARRA ESPARZA **GRUPO:** O2023_ESI3913O

ALUMNO: PAULO ALBERTO LÓPEZ RIOS **EXP:** 741876

ALUMNO: CARLOS CALZADA DIAZ **EXP:** 738803

FECHA: 01/NOVIEMBRE/2023

INDICE

FECHA: 01/NOVIEMBRE/2023

DESCRIPCION Y OBJETIVO

Introducción

El problema de las Torres de Hanói consiste en desplazar una columna hecha a base de discos concéntricos de diferentes diámetros, apilados del de mayor diámetro al de menor diámetro, desde su posición original A hasta su posición destino C,
teniendo el punto B disponible para movimientos intermedios. Dicho desplazamiento cuenta con la restricción de que solo se puede mover un disco a la vez, y que en ningún momento puede un disco de mayor diámetro estar sobre un disco de menor diámetro.

Existen varios algoritmos para su solución, pero uno de los más eficientes es el algoritmo recursivo. Este consiste en lo
siguiente:

Para mover una torre de N discos de A a C:

1. Mover los N-1 discos superiores de la torre de A a B, siendo B la posición que no corresponde ni al origen ni al destino.
2. Mover el disco restante (el más grande) de A a C.
3. Mover los N-1 discos de B a C, colocándolos encima del disco de mayor diámetro.

Como no se pueden mover N-1 en un solo movimiento, a menos que N-1 sea igual a 1, entonces se repiten los pasos
anteriores para mover los N-1 discos de A a B, y luego de B a C.

Generalmente se construye una función que recibe como parámetros el número de discos a mover (N), y las posiciones
origen y destino para ese movimiento. Si el número de discos a mover no es 1, entonces se llama recursivamente la misma
función para mover N-1 discos.

Actividades a realizar

Construir un programa en lenguaje ensamblador de RISC-V, que pueda ser ensamblado y simulado con el programa RARS, para resolver el problema de las Torres de Hanói. La solución debe ser a través de una implementación con **código recursivo**.

Durante la simulación del programa en RARS, deberá ser posible observar los movimientos de los discos en la ventana que despliega el contenido de la **memoria de datos** (data segment). Si se pone un breakpoint en el lugar adecuado del código, se deberá poder observar el movimiento de cada disco.

Entregables

Archivo .asm con la implementación de programa con las siguientes características:

FECHA: 01/NOVIEMBRE/2023

A. Código ampliamente documentado con comentarios que permitan inferir la intención de las líneas de
código de la implementación.

B. El programa debe comenzar con un encabezado con los nombres de los integrantes del equipo.

C. El programa debe soportar N discos para su ejecución.

D. Cambiando una constante que se almacena en el $s0 el programa se debe poder seleccionar el número de discos e inicializarlos en la torre orígen.

Un reporte en formato PDF que contenga:

A. Presentación con los nombres y números de expediente de los integrantes del equipo con páginas
numeradas.

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

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

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

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

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

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

H. Conclusiones de cada integrante del equipo.

El codigo funete y el reporte se deben entregar en git classroom y en canvas.

Restricciones:

1. Esta práctica deberá realizarse en equipos de 2 personas los cuales ya están formados en canvas. Excepcionalmente se aceptará que se realice individualmente (si el número de personas inscritas en un grupo es impar, una persona deberá trabajar sola).

2. La solución al algoritmo se debe implementar de **manera recursiva** (tomar como referencia la implementación de Fibonacci vista en clase). No se aceptarán prácticas que **NO** resuelvan el problema de **manera recursiva**.

3. La solución de la práctica tiene que hacer uso del **stack** para implementar los llamados recursivos, deben almacenar ra, el número de discos y los apuntadores las 3 torres.

FECHA: 01/NOVIEMBRE/2023

4. No se puede hacer equipo con integrantes de otro grupo de Organización y Arquitectura de Computadoras.
5. Se debe emplear solo las instrucciones del RV32I Base Integer Instructions. No se aceptarán prácticas que empleen otras instrucciones, o que empleen pseudo-instrucciones.

FECHA: 01/NOVIEMBRE/2023

DESARROLLO DE ACTIVIDAD

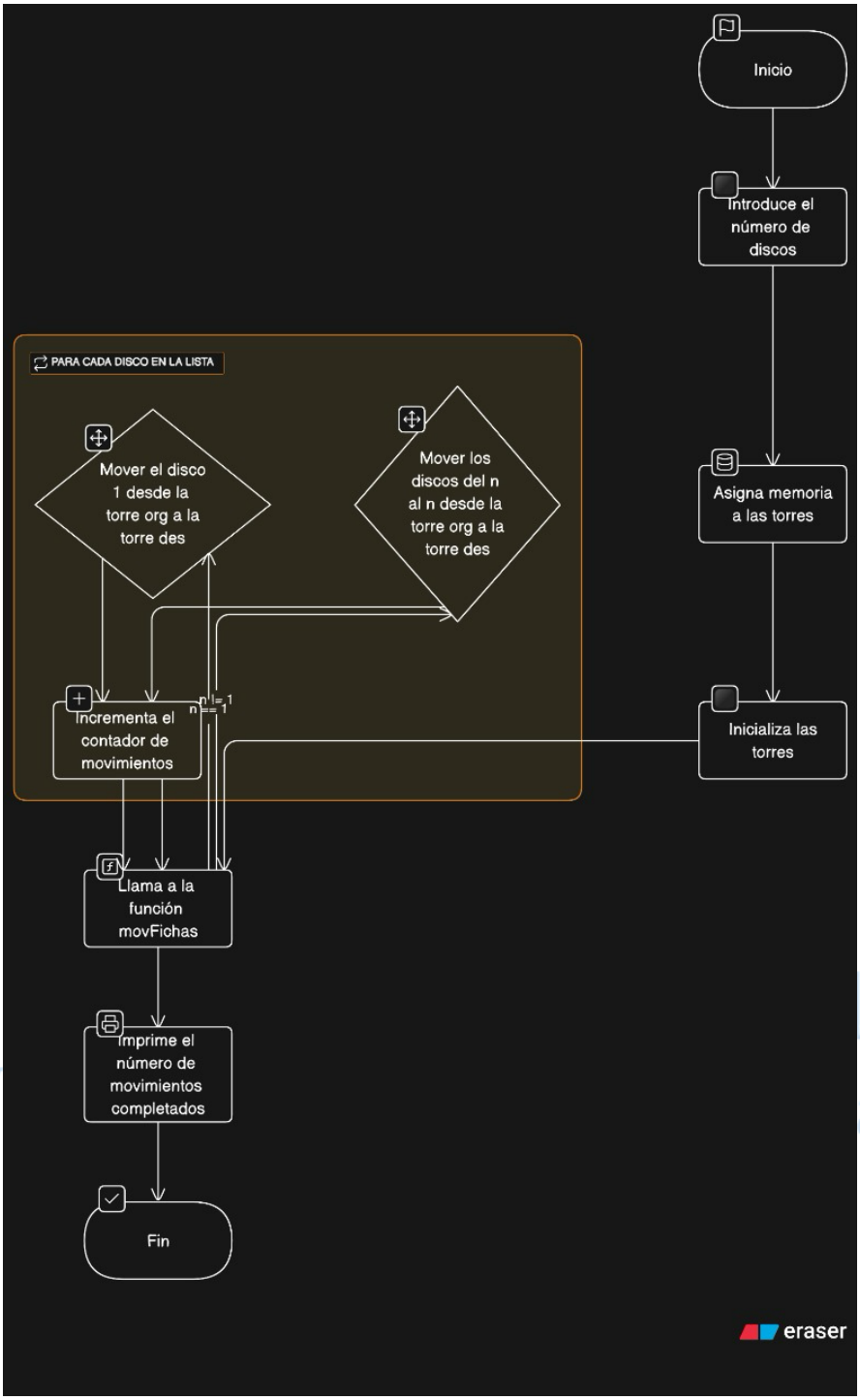


Ilustración 1 – DIAGRAMA DE FLUJO

Las Decisiones Que Se Tomaron Al Diseñar Su Programa

1. **Modelo Recursivo**: Desde las primeras discusiones en equipo, se reconoció que el problema de las Torres de Hanói tiene una estructura inherente que se presta bien

a una solución recursiva. El equipo decidió adoptar esta aproximación, no solo porque es una representación fiel del problema, sino también porque reduce la complejidad del código al evitar bucles y estructuras repetitivas innecesarias.

2. **Uso del Stack para llamadas recursivas**: Con la decisión de implementar una solución recursiva, el equipo tuvo que considerar cómo gestionar las múltiples llamadas a la función **hanoi**. Se llegó al consenso de que el uso del stack para almacenar el estado actual antes de cada llamada recursiva y restaurarlo después de cada retorno era esencial para mantener la claridad y la eficiencia del código.

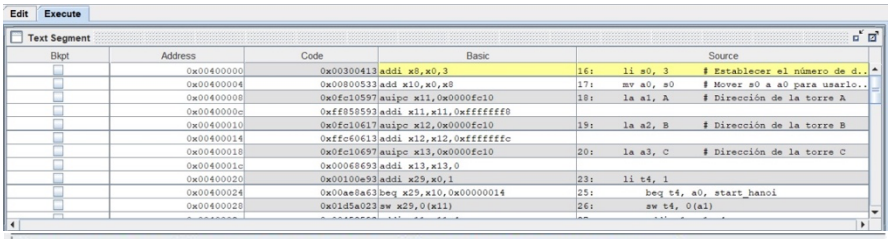
3. **Representación de las Torres**: Durante las sesiones de brainstorming, el equipo consideró varias representaciones posibles para las torres. Al final, se decidió que representar las torres como áreas contiguas de memoria sería la más directa y eficiente. Las operaciones en las torres se realizan utilizando instrucciones básicas de carga y almacenamiento, lo que facilita su implementación y depuración.

4. **Inicialización de las Torres**: El equipo decidió que, para simplificar la lógica, la torre A se inicializaría con todos los discos al comienzo del programa, mientras que las torres B y C estarían vacías. Esto establece claramente las condiciones iniciales del problema y permite que el programa se concentre en la lógica de mover los discos.

5. **Movimiento de Discos**: Al discutir cómo implementar el movimiento de discos entre torres, el equipo decidió encapsular esta lógica en una función separada **move_disk**. Esto no solo facilita la legibilidad del código, sino que también permite posibles optimizaciones y modificaciones sin afectar la lógica principal del programa.

6. **Decisiones de Flujo**: El equipo pasó tiempo discutiendo cómo gestionar las decisiones de flujo en el programa. Se decidió que, en lugar de usar una lógica condicional compleja, las decisiones se basarían en el valor del número de discos **n**. Si **n** es 1, se mueve un disco. Si es mayor que 1, se realiza una llamada recursiva para mover **n-1** discos.

7. **Optimización del Flujo**: Una innovación clave propuesta por un miembro del equipo fue la idea de intercambiar las direcciones de las torres para evitar lógicas condicionales complejas. Esta decisión simplificó enormemente el código y redujo el potencial de errores.



Registers | Floating Point | Control and Status

Name	Number	Value
ustatus	0	0x00000000
fflags	1	0x00000000
frm	2	0x00000000
fcsr	3	0x00000000
uie	4	0x00000000
utvec	5	0x00000000
uscratch	64	0x00000000
uepc	65	0x004000bc
ucause	66	0x00000007
utval	67	0x0ffffffc
uip	68	0x00000000
cycle	3072	0x0009c15d
time	3073	0x8e4b1087
instret	3074	0x0009c15d
cycleh	3200	0x00000000
timeh	3201	0x0000018b
instreth	3202	0x00000000

Analisis Del Comportamiento

1. **Contexto inicial**:
   - Al principio, el stack está vacío y esperando cualquier operación.
   - Se toma la decisión de mover 3 discos de la torre A a la torre C, usando la torre B como auxiliar. Esta es la operación principal y el objetivo general del programa.
2. **Primera llamada a hanoi con 3 discos**:
   - Antes de sumergirnos en la recursión para mover los 2 discos superiores, necesitamos guardar nuestro lugar actual en la ejecución. Esto implica almacenar en el stack el registro de retorno **ra**, que nos dice dónde reanudar una vez que hayamos terminado con la llamada recursiva. También almacenamos el número de discos (3 en este caso) y las direcciones de las tres torres para recordar nuestro contexto.

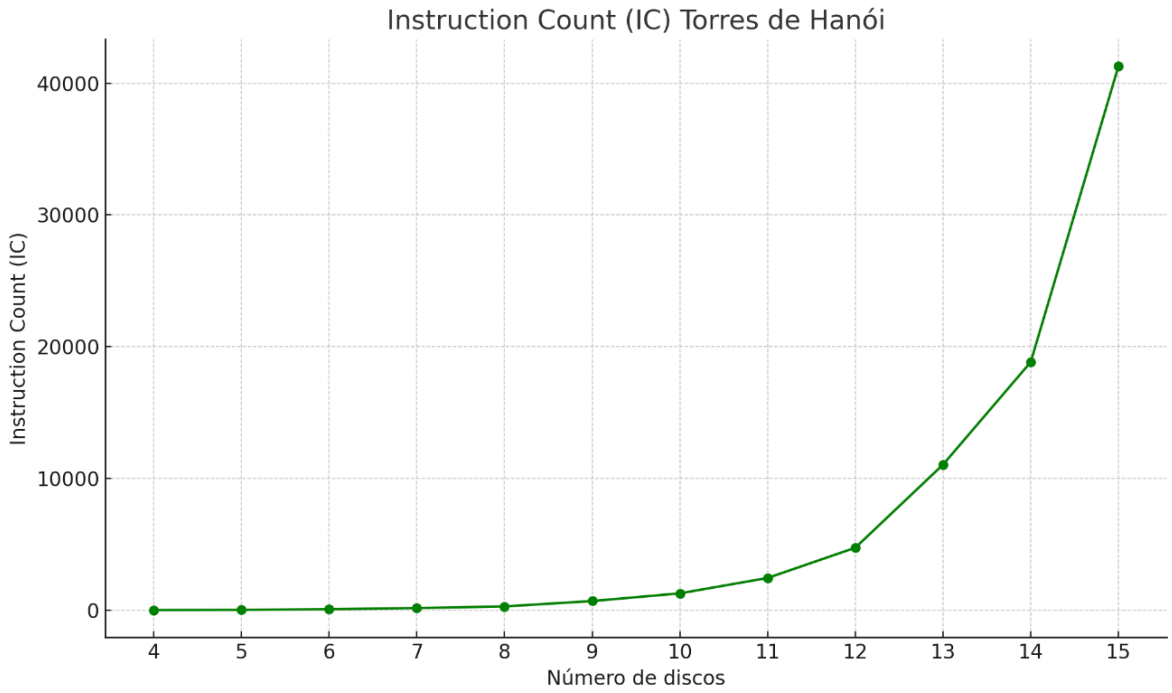
FECHA: 01/NOVIEMBRE/2023

- Con nuestro estado actual guardado, procedemos a la primera llamada recursiva. La tarea ahora es mover 2 discos de la torre A a la torre B, usando la torre C como intermediario.

3. **Segunda llamada a hanoi con 2 discos**:
   - Al igual que antes, queremos recordar nuestro lugar en la ejecución. Guardamos el estado actual en el stack, que incluye el registro de retorno, el número de discos (2 esta vez) y las direcciones de las torres.
   - Con este estado guardado, nos embarcamos en otra llamada recursiva. La misión es mover 1 disco de la torre A a la torre C, con la torre B como auxiliar.
4. **Tercera llamada a hanoi con 1 disco**:
   - Ahora, la recursión nos ha llevado al caso base. Cuando solo hay 1 disco, no necesitamos más recursión. Simplemente movemos el disco de la torre origen a la torre destino.
   - Una vez que hemos movido este disco, es hora de regresar a nuestra operación anterior. Usamos el stack para recuperar el estado de la segunda llamada, recordando dónde estábamos y qué estábamos haciendo.
5. **Retorno a la segunda llamada**:
   - Después de mover el disco más grande de la torre A a la torre C, todavía tenemos una tarea pendiente: mover los 2 discos restantes de la torre B a la torre C. Sin embargo, ahora la torre A está libre, así que la usamos como auxiliar.
   - Una vez que hemos completado esta operación, es hora de regresar a la primera llamada. Utilizamos el stack nuevamente para restaurar el estado y recordar el contexto de la primera llamada.
6. **Retorno a la primera llamada**:
   - Con todos los discos movidos correctamente de la torre A a la torre C, hemos logrado nuestro objetivo. Sin embargo, todavía necesitamos asegurarnos de que cualquier estado restante en el stack se restaure correctamente para que el programa concluya de manera ordenada.
   - Finalmente, una vez que todo el estado se ha restaurado y todas las operaciones se han completado, el programa finaliza.

A través de este proceso detallado, podemos apreciar la importancia del stack en la gestión de la recursión y cómo se utiliza para guardar y restaurar el estado a medida que el

programa se sumerge en llamadas recursivas y luego regresa de ellas. Es esencial manejar el stack correctamente para garantizar que el programa funcione como se espera.



CONCLUSIONES

Paulo Alberto Lopez Rios: El reto de implementar las Torres de Hanói en ensamblador RISC-V ha sido una de mis experiencias más formativas en programación. No solo me ha permitido profundizar en la intrincada naturaleza de la programación a nivel de ensamblador, sino que también ha resaltado la elegancia y complejidad de los problemas recursivos. A medida que avanzábamos, la necesidad de colaborar en equipo y aunar esfuerzos se volvía cada vez más evidente, siendo fundamental para superar los obstáculos que encontramos en el camino. Esta experiencia ha reforzado mi convicción sobre la importancia de la optimización de algoritmos, especialmente cuando enfrentamos problemas con una complejidad creciente. La satisfacción de ver el problema resuelto, y el conocimiento adquirido en el proceso, ha sido invaluable.

Carlos Esteban Calzada Diaz: Abordar un problema tan clásico como las Torres de Hanói desde la perspectiva del ensamblador RISC-V ha sido una travesía que ha enriquecido enormemente mi comprensión técnica. A través de este desafío, he podido apreciar la profundidad y precisión requerida al trabajar con estructuras de bajo nivel en programación, algo que a menudo se da por sentado en lenguajes de alto nivel. La gestión