INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática

ORGANIZACIÓN Y ARQUITECTURA DE COMPUTADORAS

ITESO, Universidad Jesuita de Guadalajara

< PRACTICA 1 >

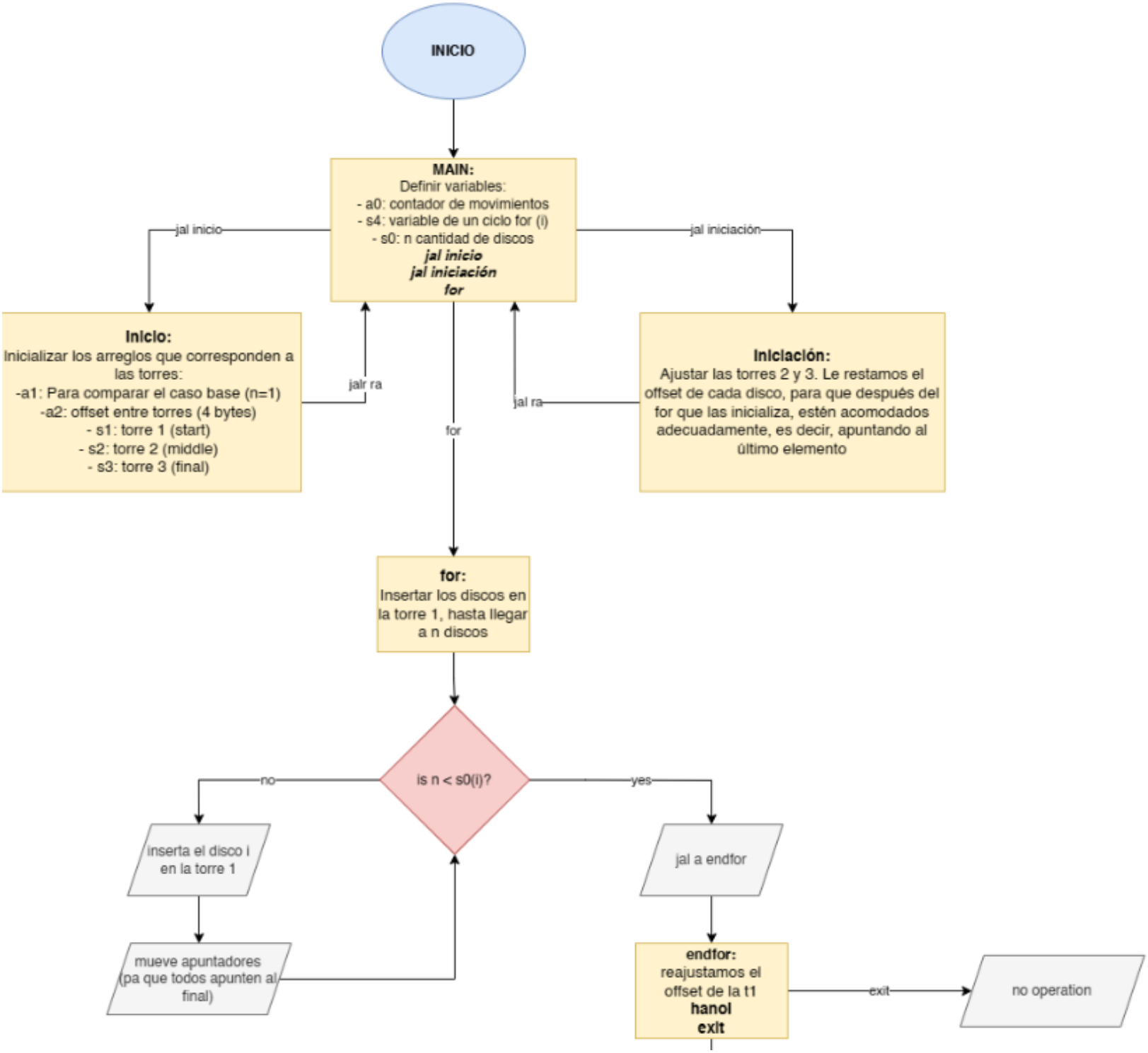
Presenta
Angel Gabriel Ramírez Carrillo
José Juan Diaz Campos

Profesora: Ibarra Esparza, Juanpablo
Fecha: <17 03 2024>

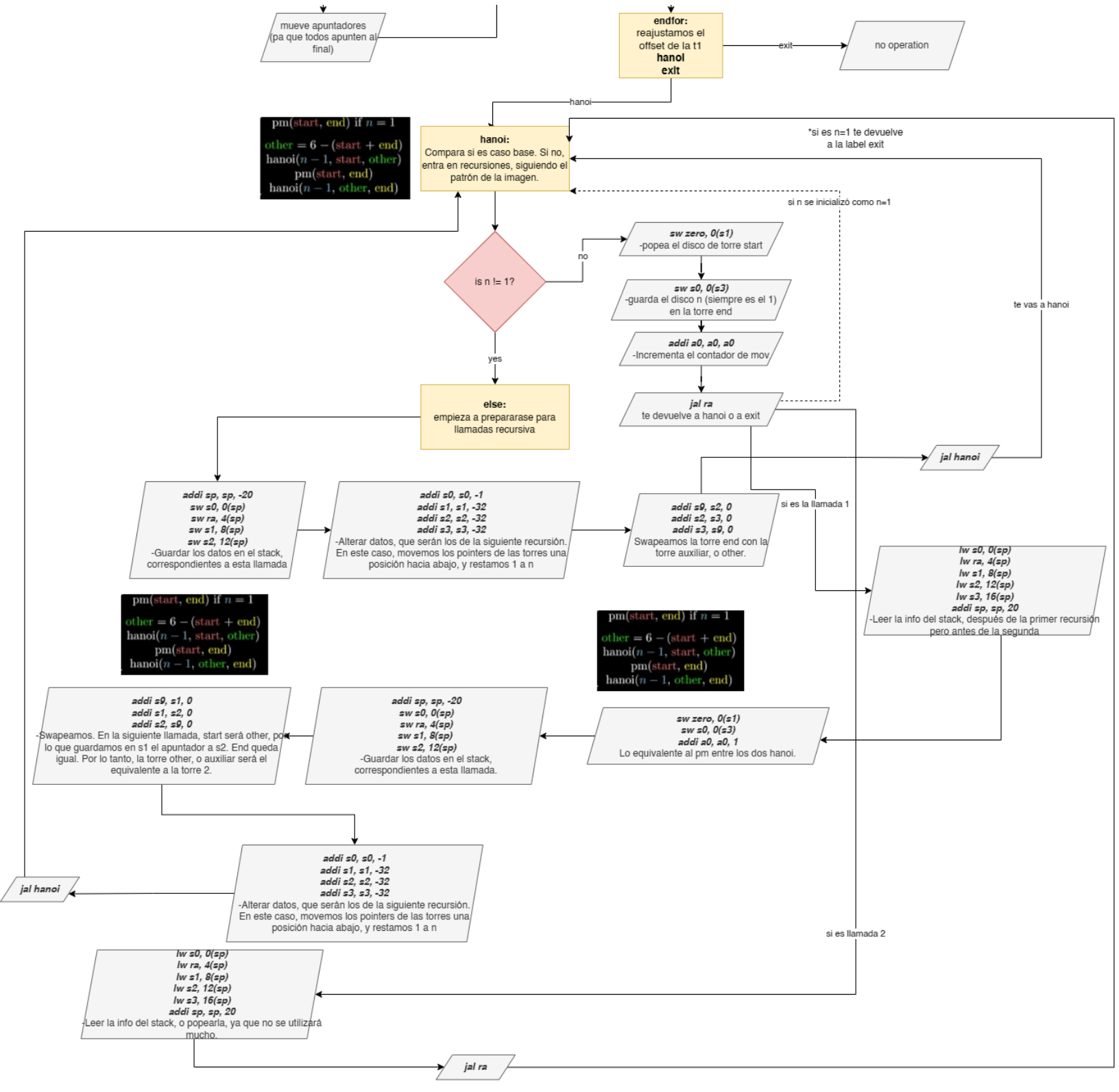
Contenido

Diagrama de flujo

PT1



PT2


mueve apuntadores (pa que todos apunten al final)
endfor:
reajustamos el offset de la t1
hanoi
exit
exit
no operation
hanoi
pm(start, end) if n = 1
other = 6 − (start + end)
hanoi(n − 1, start, other)
pm(start, end)
hanoi(n − 1, other, end)
hanoi:
Compara si es caso base. Si no, entra en recursiones, siguiendo el patrón de la imagen.
*si es n=1 te devuelve a la label exit
si n se inicializó como n=1
is n != 1?
no
sw zero, 0(s1)
-popea el disco de torre start
sw s0, 0(s3)
-guarda el disco n (siempre es el 1) en la torre end
addi a0, a0, a0
-Incrementa el contador de mov
jal ra
te devuelve a hanoi o a exit
te vas a hanoi
yes
else:
empieza a prepararse para llamadas recursiva
jal hanoi
addi sp, sp, -20
sw s0, 0(sp)
sw ra, 4(sp)
sw s1, 8(sp)
sw s2, 12(sp)
-Guardar los datos en el stack, correspondientes a esta llamada
addi s0, s0, -1
addi s1, s1, -32
addi s2, s2, -32
addi s3, s3, -32
-Alterar datos, que serán los de la siguiente recursión. En este caso, movemos los pointers de las torres una posición hacia abajo, y restamos 1 a n
addi s9, s2, 0
addi s2, s3, 0
addi s3, s9, 0
Swapeamos la torre end con la torre auxiliar, o other.
si es la llamada 1
lw s0, 0(sp)
lw ra, 4(sp)
lw s1, 8(sp)
lw s2, 12(sp)
lw s3, 16(sp)
addi sp, sp, 20
-Leer la info del stack, después de la primer recursión pero antes de la segunda
pm(start, end) if n = 1
other = 6 − (start + end)
hanoi(n − 1, start, other)
pm(start, end)
hanoi(n − 1, other, end)
pm(start, end) if n = 1
other = 6 − (start + end)
hanoi(n − 1, start, other)
pm(start, end)
hanoi(n − 1, other, end)
addi s9, s1, 0
addi s1, s2, 0
addi s2, s9, 0
-Swapeamos. En la siguiente llamada, start será other, por lo que guardamos en s1 el apuntador a s2. End queda igual. Por lo tanto, la torre other, o auxiliar será el equivalente a la torre 2.
addi sp, sp, -20
sw s0, 0(sp)
sw ra, 4(sp)
sw s1, 8(sp)
sw s2, 12(sp)
-Guardar los datos en el stack, correspondientes a esta llamada.
sw zero, 0(s1)
sw s0, 0(s3)
addi a0, a0, 1
Lo equivalente al pm entre los dos hanoi.
addi s0, s0, -1
addi s1, s1, -32
addi s2, s2, -32
addi s3, s3, -32
-Alterar datos, que serán los de la siguiente recursión. En este caso, movemos los pointers de las torres una posición hacia abajo, y restamos 1 a n
jal hanoi
si es llamada 2
lw s0, 0(sp)
lw ra, 4(sp)
lw s1, 8(sp)
lw s2, 12(sp)
lw s3, 16(sp)
addi sp, sp, 20
-Leer la info del stack, o popearla, ya que no se utilizará mucho.
jal ra

Decisiones tomadas

Intentamos hacer el código lo más legible posible puesto que de por si es algo difícil de entender el ensamblador, intentamos encapsular todo lo posible en labels, esto nos ayudó a trabajar de una forma más fácil y también presenta una mejor lectura, quisimos hacer lo mismo en el else en las partes que se trabaja con el stack pero por alguna razón fallaba, para el conocimiento que presentamos en este momento no fuimos capaces de resolverlo y optamos por dejarlo así, la razón por la cual el inicio e inicialización están hasta bajo es que no son cosas tan pertinentes de observar en la ejecución del código, y cuando estábamos trabajando era más fácil saltar a lo que necesitaba trabajo de esta forma.

Lo hicimos vertical porque, básicamente, nadie que conocíamos lo había probado aún, y es mucho más legible de esa manera. Eso sí, no pudimos replicar el comportamiento de los números para que parecieran discos, debido a que, para tener control de los apuntadores, y no complicarnos, siempre están apuntando a las mismas posiciones, en sus respectivas filas. Esto quiere decir, que cuando movemos el disco 1, 2 o 3 a alguna torre, estarán en esas posiciones al moverse.

Esto no significa que se rompan las reglas del algoritmo, pero si impacta visualmente, pues no replica un comportamiento realista de las torres, aunque sea el correcto (no hay discos grandes arriba de los chiquitos).

Fuera de eso, todo marchó con normalidad.

Simulación en RARS para 3 discos

Inicio:

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	1	0	0
0x10010020	2	0	0
0x10010040	3	0	0
0x10010060	0	0	0
0x10010080	0	0	0
0x100100a0	0	0	0
0x100100c0	0	0	0
0x100100e0	0	0	0
0x10010100	0	0	0
0x10010120	0	0	0
0x10010140	0	0	0
0x10010160	0	0	0
0x10010180	0	0	0
0x100101a0	0	0	0

Paso 1 -> Disco 1 a end

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0	0	1
0x10010020	2	0	0
0x10010040	3	0	0
0x10010060	0	0	0

Paso 2 -> Disco 2 a middle

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0	0	1
0x10010020	0	2	0
0x10010040	3	0	0
0x10010060	0	0	0

Paso 3 -> Disco 1 a middle

Data Segment

Address	Value (+0)	Value (+4)
0x10010000	0	1
0x10010020	0	2
0x10010040	3	0
0x10010060	0	0

Paso 4 -> Disco 3 a end

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0	1	0
0x10010020	0	2	0
0x10010040	0	0	3
0x10010060	0	0	0

Paso 5 -> Disco 1 a start

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	1	0	0
0x10010020	0	2	0
0x10010040	0	0	3
0x10010060	0	0	0
0x10010080	0	0	0

Paso 6 -> Disco 2 a end

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	1	0	0
0x10010020	0	0	2
0x10010040	0	0	3

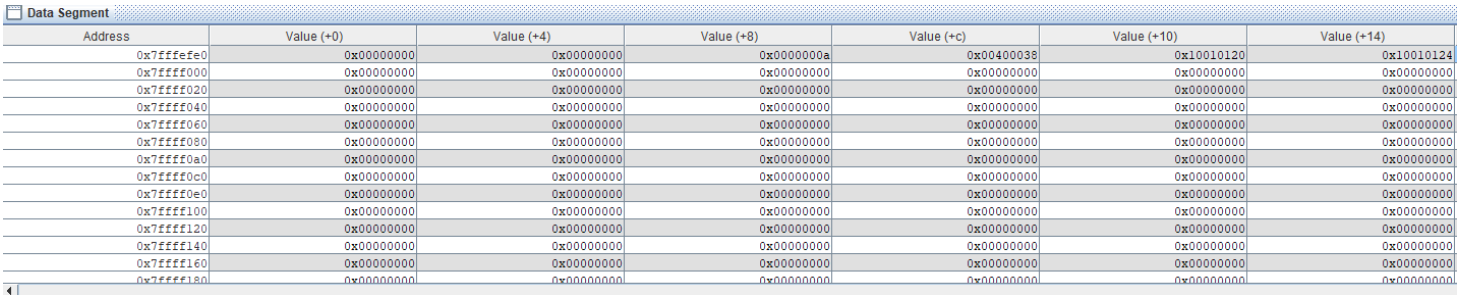
Paso 7 -> Disco 1 a end

Address	Value (+0)	Value (+4)	Value (+8)
0x10010000	0	0	1
0x10010020	0	0	2
0x10010040	0	0	3
0x10010060	0	0	0

Análisis

Inicializamos los discos utilizando un ciclo for, lo que nos permite manejar tanto el caso específico de 3 discos como cualquier número n de discos. Esto nos brinda una solución escalable y generalizada para el problema de la Torre de Hanoi, independientemente de la cantidad de discos involucrados.

El uso del stack es fundamental para permitir la recursión, ya que nos permite preservar los datos necesarios en cada llamada recursiva y dar seguimiento a los movimientos realizados. Cuando se realiza una llamada recursiva, el estado actual del problema, incluyendo la configuración de los discos y las torres, se almacena en el stack



Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)
0x7fffefe0	0x00000000	0x00000000	0x0000000a	0x00400038	0x10010120	0x10010124
0x7ffff000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff0e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff100	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff120	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff140	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff160	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x7ffff180	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

Ejemplo: n=10. En la primera entrada a la recursión, guarda los apuntadores al fondo de los arreglos, y el valor de n, que en ese momento es 10. La próxima vez se utilizarán para el resto de cálculos.

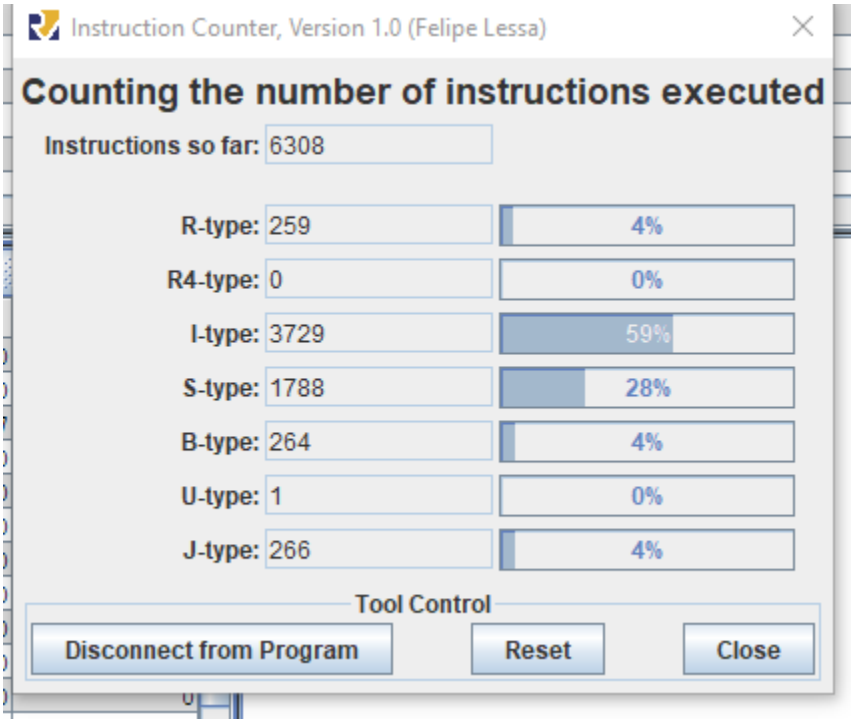
```
addi sp, sp, -20        # 1ra llamada a la funcion hanoi (Guarda los registros en la pila)
sw s0, 0(sp)            #valor de n en esta llamada
sw ra, 4(sp)            #ra, para volver
sw s1, 8(sp)            #apuntador a la torre 1
sw s2, 12(sp)           #apuntador a la torre 2
sw s3, 16(sp)           #apuntador a la torre 3
```

Guardamos los datos…

```
lw s0, 0(sp)            # Restaura los registros de la pila
lw ra, 4(sp)
lw s1, 8(sp)
lw s2, 12(sp)
lw s3, 16(sp)
addi sp, sp, 20
```

¡Y los sacamos de vuelta! Acomodando el pointer del stack.

Instruction count



De las cuales tipo R son 259 o el 4% tipo i son 3729 o el 59% y tipo j son 266 o el 4%

<u>Grafica que muestra el comportamiento</u>

numero de discos	IC
4	400
5	799
6	1590
7	3165
8	9473
9	12587
10	25138
11	50233
12	100416
13	200775
14	401486
15	802901

Conclusiones

Ángel:

Sinceramente fue un proyecto que si tenía su nivel de complejidad, jamás en mi vida había programado en assembly, tenía una noción de lo difícil que podía resultar ser, pero no tenia en cuenta que iba ser así, sin embargo gracias a la orientación que nos dio el profesor sobre como podíamos hacer el movimiento de las torres es que me quedo mucho más claro cómo es que se manejaba, agradezco mucho que el proyecto se pudo realizar entre 2 personas, pues esto mermo considerablemente la dificultad de la actividad, creo que el mejor aprendizaje que obtengo de esta práctica es lo complejo pero eficiente que puede ser trabajar con ensamblador.

José Juan:

El proyecto estuvo cañón. Pero nos dejó muchísima práctica, y hasta eso, es una pena que no sigamos practicando más assembly en RISC V. Creo que sería interesante seguir navegando el mundo que tiene detrás. Creo que el mayor reto fue entender el código recursivo de hanoi primero, para considerar implementarlo posteriormente. Una vez hecho eso, fue más sencilla la práctica. No significa que fuese sencilla, solo pasó de ser imposible a un proyecto difícil.