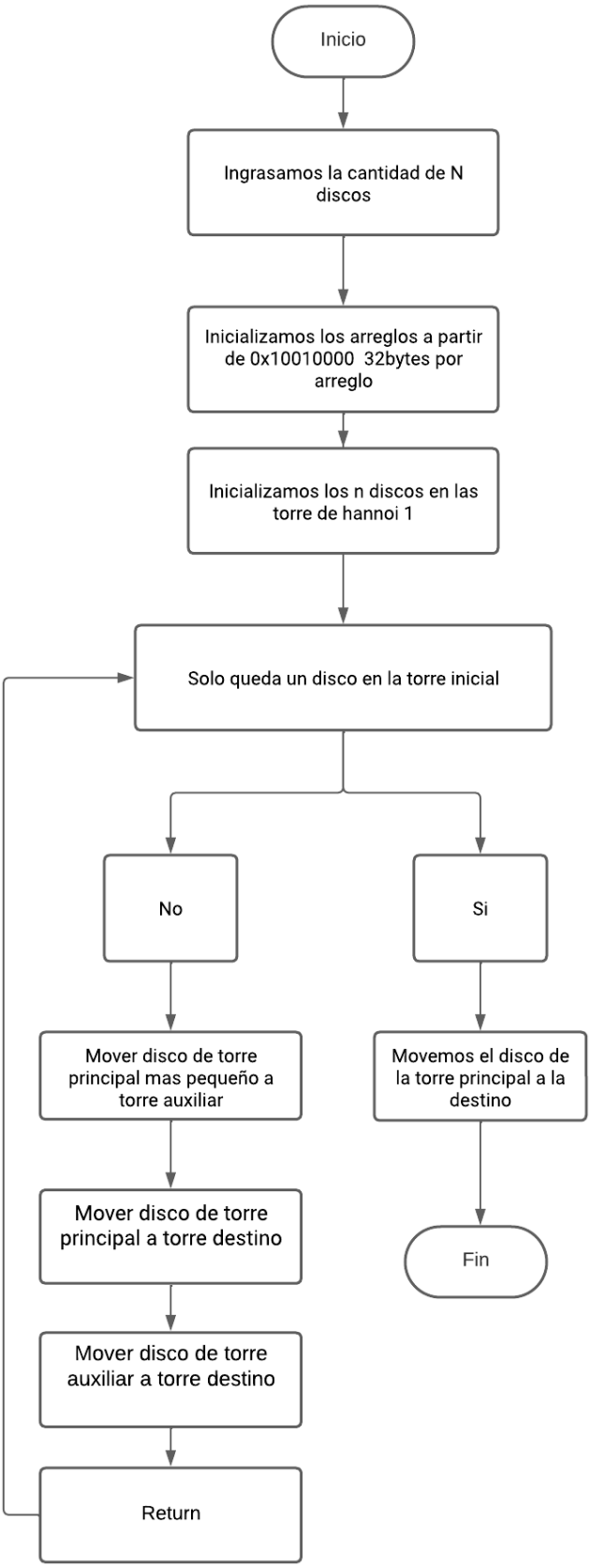
ITESO, Universidad Jesuita de Guadalajara

Practica 1

Torres de Hannoi

Luis Miguel Fontes Cardona - 741547

Victor Sebastian Huerta Silva – 740347


Inicio
Ingrasamos la cantidad de N discos
Inicializamos los arreglos a partir de 0x10010000 32bytes por arreglo
Inicializamos los n discos en las torre de hannoi 1
Solo queda un disco en la torre inicial
No
Si
Mover disco de torre principal mas pequeño a torre auxiliar
Movemos el disco de la torre principal a la destino
Mover disco de torre principal a torre destino
Fin
Mover disco de torre auxiliar a torre destino
Return

Decisiones que se tomaron al diseñar su programa:

El código que utilizamos como referencia fue este:

```c
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    if (n == 1)
    {
        printf("\n Move disk 1 from rod %c to rod %c", from_rod, to_rod);
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("\n Move disk %d from rod %c to rod %c", n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}

int main()
{
    int n = 4; // Number of disks
    towerOfHanoi(n, 'A', 'C', 'B');  // A, B and C are names of rods
    return 0;
}
```

Para comenzar, primero tuvimos que definir los registros que íbamos a utilizar. Estas variables incluyen la inicialización de la RAM (puntero al arreglo). Logramos esto mediante un "lui", estableciendo la referencia a la memoria RAM con el valor 0x10010. A partir de este punto, creamos referencias a las otras torres o arreglos mediante una multiplicación que nos permitió calcular los saltos. Para ello, empleamos un "slli" (Shift Left Logical Immediate) con un valor de 2, multiplicando así por 4 el número de discos, que corresponde al offset de la memoria RAM. Luego, sumamos la dirección de s1 (nuestra primera dirección RAM) con el resultado de nuestro offset, obteniendo así las referencias a los arreglos de las torres A, B y C.

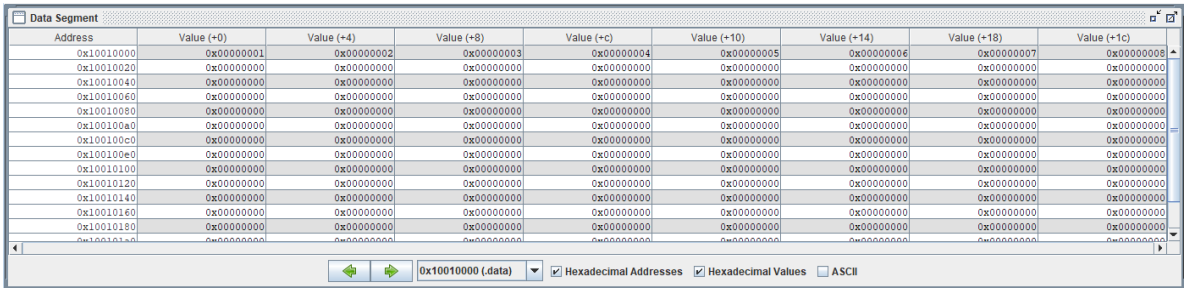
```
.text
        addi t0, zero, 1        # i = 1 (ciclo for)
        addi t1, zero, 1        # Inicializacion en 1 para el caso default
        addi a2, zero, 8        # n = 8 (8 discos)

        slli s0, a2, 2          # Multiplicar por 4

        lui s1, 0x10010         # Apuntador al Arreglo A
        add s2, s1, s0          # Apuntador al Arreglo B
        add s3, s2, s0          # Apuntador al Arreglo C
```

Ahora necesitamos generar los valores de las torres. Logramos esto mediante un bucle "for", en el cual recorremos utilizando un registro temporal (para no perder la referencia a nuestra RAM) y almacenamos el valor de la variable "i" del ciclo "for". Al estar inicializada en 1 y aumentar con "i++", nos permite definir gradualmente los valores en nuestra torre.

```
for:                            # Ciclo for para inicializar los valores de la torre
        blt a2, t0, endfor      # Condicion del ciclo for: i < n
        sw t0, 0(s1)            # Inicializamos el valor de i en la torre A
        addi s1, s1, 4          # Nos movemos 4 bits para la siguiente posicion
        addi s2, s2, 4
        addi s3, s3, 4
        addi t0, t0, 1          # i++
        jal for
```



Antes de comenzar con la función de “towerOFHanoi”. Tenemos que regresar los apuntadores a su última posición. Esto gracias a que el ciclo for recorre una vez más los apuntadores.

```
endfor:
        addi s1, s1, -4         # Como el ciclo for se ejecuta 1 vez mas de lo necesario
        addi s2, s2, -4         # Se tiene que restar -4 para no perder la referencia al
        addi s3, s3, -4         # ultimo espacio correspondiente de la torre

        jal towerOfHanoi
        jal endcode
```

Para la función 'towerOfHanoi', necesitamos dividir nuestros caminos en los casos predeterminado y base, identificados como el caso default y el caso 0, respectivamente. Para esto, empleamos una instrucción 'beq' para comparar si el número de discos es igual a 1. Si esto no ocurre, llevamos a cabo un 'push' en el Puntero de Pila con el número de discos (n), las tres torres (A, B y C) y ra (nuestra variable de retorno). Luego, se ajustan los argumentos, disminuyendo n en 1 para reducir la cantidad de discos, recorriendo los arreglos y realizando el intercambio entre las torres correspondientes. Finalmente, ejecutamos un 'pop' para extraer los valores. Es importante destacar que al realizar el 'push', debemos reservar espacio en el Puntero de Pila restando 4; en contraste, para efectuar el 'pop', debemos añadir 4.

```
towerOfHanoi:
        beq a2, t1, default     # if n == 1 -> default

        # Necesitamos crear espacio en el SP (Push al stack), para esto restamos -4 al SP
        addi sp, sp, -4
        sw ra, 0(sp)            # Metemos el return (ra)
        addi sp, sp, -4
        sw s1, 0(sp)            # Metemos la Primera torre (A)
        addi sp, sp, -4
        sw s2, 0(sp)            # Metemos la Segunda torre (B)
        addi sp, sp, -4
        sw s3, 0(sp)            # Metemos la Tercera torre (C)
        addi sp, sp, -4
        sw a2, 0(sp)            # Metemos el numero de discos (n)

        # Modificacion de argumentos
        addi a2, a2, -1         # n - 1 de los discos

        addi s1, s1, -4         # Recorremos los apuntadores
        addi s2, s2, -4
        addi s3, s3, -4

        # Switch de las torres
        addi t3, s2, 0          # Asignacion de s2 a un auxiliar
        addi s2, s3, 0          # Switch de s2 con s3
        addi s3, t3, 0          # Switch de s3 con auxiliar

        jal towerOfHanoi        # Llamada recursiva

        # Pop del stack
        lw a2, 0(sp)
        addi sp, sp, 4
        lw s3, 0(sp)
        addi sp, sp, 4
        lw s2, 0(sp)
        addi sp, sp, 4
        lw s1, 0(sp)
        addi sp, sp, 4
        lw ra, 0(sp)
        addi sp, sp, 4
```

Realizamos el switch/swap de discos, modificando la torre inicial con 0 y reescribiéndolo en la torre correspondiente.

```
# Hacemos el swap de discos
sw zero, 0(s1)          # Pop al disco
sw a2, 0(s3)            # Push al disco
```

En nuestro código en C, se nota que la recursión de la función "towerOfHanoi" se ejecuta dos veces. Por lo tanto, necesitamos repetir la ejecución de esta función, guardando nuevamente los valores con un 'push' en el Puntero de Pila, ajustando los valores, invocando la función y, finalmente, realizando un 'pop'.

```
# Necesitamos crear espacio en el SP (Push al stack), para esto restamos -4 al SP
addi sp, sp, -4
sw ra, 0(sp)            # Metemos el return (ra)
addi sp, sp, -4
sw s1, 0(sp)            # Metemos la Primera torre (A)
addi sp, sp, -4
sw s2, 0(sp)            # Metemos la Segunda torre (B)
addi sp, sp, -4
sw s3, 0(sp)            # Metemos la Tercera torre (C)
addi sp, sp, -4
sw a2, 0(sp)            # Metemos el numero de discos (n)

# Modificacion de argumentos
addi a2, a2, -1         # n - 1 de los discos

addi t2, s1, 0          # Asignacion de s1 a un auxiliar
addi s1, s2, 0          # Switch de s1 con s2
addi s2, t2, 0          # Switch de s2 con auxiliar

addi s1, s1, -4         # Recorremos los apuntadores
addi s2, s2, -4
addi s3, s3, -4

jal towerOfHanoi        # Llamada recursiva

# Necesitamos crear espacio en el SP (Push al stack), para esto restamos -4 al SP
lw a2, 0(sp)
addi sp, sp, 4
lw s3, 0(sp)
addi sp, sp, 4
lw s2, 0(sp)
addi sp, sp, 4
lw s1, 0(sp)
addi sp, sp, 4
lw ra, 0(sp)
addi sp, sp, 4

jalr ra                 # Retornamos el valor
```

Finalmente, tenemos nuestro caso default. En donde tenemos que guardar en un registro auxiliar el valor retornado para posteriormente hacer el cambio de discos y retornar nuestro valor.

```
default:
        add t3, zero, ra        # Guardamos el valor de ra

        # Hacemos el swap de discos
        sw zero, 0(s1)          # Pop al disco
        sw a2, 0(s3)            # Push al disco

        add ra, zero, t3        # Se suma en ra

        jalr ra                 # Retorno del valor
```

Con esto tenemos comentado el funcionamiento de nuestro código.

Una simulación en el RARS para 3 discos:

Inicializamos los arreglos:

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
0x100100e0	0	0	0	0	0	0	0	0

Movemos nuestro disco más pequeño a la torre destino:

Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0	2	3	0	0	0	1	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Movemos disco 2 al auxiliar, movemos disco 1 al auxiliar:

Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0	0	3	1	2	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

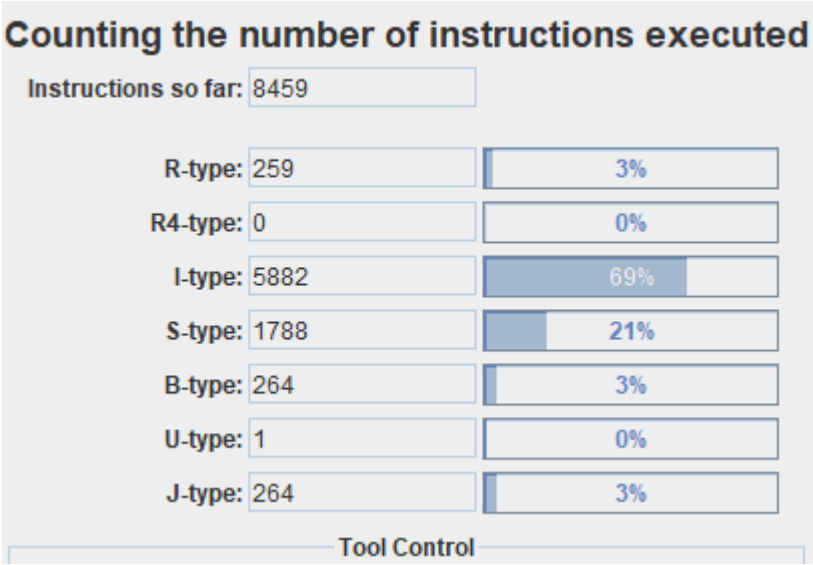
Movemos disco 3 al destino, disco 1 al auxiliar:

Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
1	0	0	0	2	0	0	0
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Movemos disco 2 al destino, movemos disco 1 al destino:

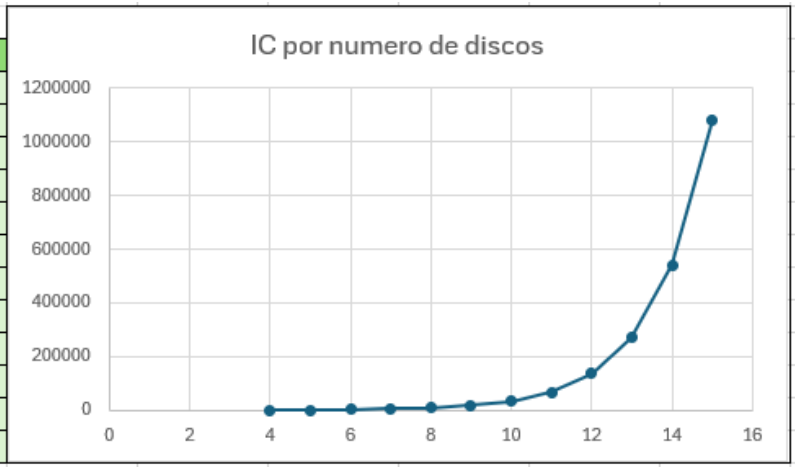
Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0	0	0	0	0	0	1	2
3	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

IC con 8 discos:



IC de 4 a 15 discos

Discos	IC
4	511
5	1046
6	2109
7	4228
8	8459
9	16914
10	33817
11	67616
12	135207
13	270382
14	540725
15	1081915



Conclusiones:

Luis Miguel Fontes Cardona:

El aprendizaje del ensamblador amplía la comprensión del funcionamiento de los sistemas informáticos. Al ser un lenguaje de bajo nivel, da un control detallado del hardware y el software, lo que no se experimenta con lenguajes de alto nivel como Python o C. Este control permite optimizar el código y depurar problemas a nivel de máquina. Además, el conocimiento de ensamblador es fundamental para comprender mejor la arquitectura de la computadora y sus componentes internos, lo que facilita el desarrollo de software eficiente y seguro. Por último, aprender ensamblador perm hacia el núcleo de la informática, ofreciendo una perspectiva más completa y una capacidad mejorada para resolver desafíos tecnológicos.

Víctor Sebastián Huerta Silva:

Considero que esta práctica sirvió bastante para comprender la manera en que nos podemos mover a través de la memoria además de pudimos practicar bastante lo de push y pop y aplicar de manera más eficiente lo de cuidar que registro es que, debido que ante tanta variable con un diferenciación de nombre de un solo digito sino lo comenta uno se puede llegar a confundir, y que es importante comprender los conceptos básicos, los cuales nunca creí que tuviera que volver aprender tales como el simple hecho de definir un variable, o hacer una operación matemática, personalmente se me complico entender cómo funcionaba el código sin verlo en consola sin ningún print pero al final logre aprender bastante gracias a esta práctica en cuestiones muy básicas que creí que ya tenía dominadas y me percate que todavía me falta.

Link del Repo: https://github.com/lmfontes19/Practica1-Arquitectura_de_Computadoras.git