PRACTICA 2

Arquitectura Computacional | Pedro Saldaña Zepeda | O. 2019

Eduardo Ethandrake Castillo Pulido
Alejandro Gudiño Gallegos
le714410@iteso.mx

Arquitectura de computadoras

Práctica 2

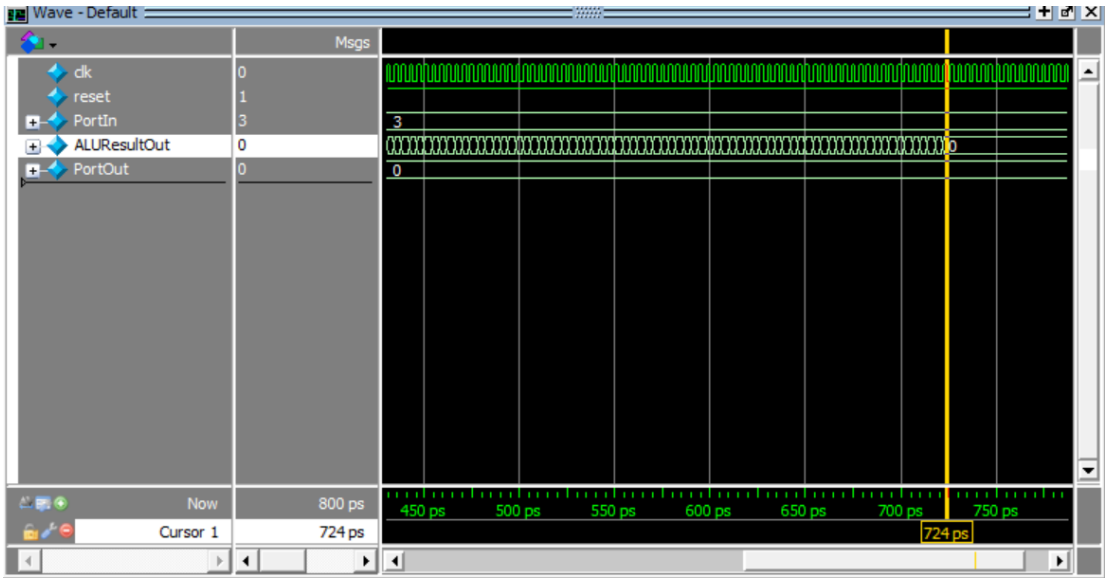
Diseño y simulación de un procesador uni-ciclo basado en la arquitectura MIPS

Actividad a realizar

Implementar en Verilog un procesador basado en la arquitectura MIPS, el cual sea capaz de ejecutar las instrucciones add, addi, sub, or, ori, and, andi, lui, nor, sll, srl, lw, sw, beq, bne, j, jal, jr las cuales se deben apegar a la especificación del MIPS Green Sheet. Dicha implementación puede usar como punto de partida el data-path visto en clase, el cual fue diseñado para soportar algunas de las instrucciones anteriores. Esta implementación debe ser capaz de ejecutar el programa que se utilizó en la practica 1, note que esta implementación debe soportar funciones recursivas. El módulo del MIPS_Processor debe emplear solo Verilog estructural, es decir, crear instancias de los módulos básicos, tales como la ALU o el register file y conectar dichas instancias. Las definiciones de los módulos básicos se encuentran en el archivo BasicMIPSProcessor.rar, el cual se puede descargar desde Moodle. Se puede emplear el comando assign dentro del módulo MIPS_Processor para definir buses especiales o crear salidas en el top-level. Para poder ejecutar las pruebas se necesita que estén disponibles el archivo text.dat en el mismo subdirectorio que el proyecto de Modelsim. Este archivo contiene las instrucciones que se cargarán en la memoria de programa. El archivo text.dat se puede generar usando el programa MARS, ensamblando un programa de prueba y exportando los datos del segmento .text (segmento de programa) en formato texto hexadecimal. En Modelsim debe poder observarse la ejecución individual de cada instrucción a través de las distintas señales (buses y señales de control) que conforman al diseño. Además, el valor de PC (0x400000) y el address (0x10010000) de la memoria de datos deben coincidir con la simulación en el MARS.

El IC es de 6034, del cual 259 son de tipo R, 5390 son de tipo I y 384 son de tipo J. Lo que quiere decir que el 4% de las instrucciones son de tipo R, el 89% de las instrucciones son de tipo I y el 6% son de tipo J. El clock rate es de 42.26MHz. El CPI del MIPS es de 1 ciclo por instrucción, esto es porque el MIPS que se diseño es uni-ciclo, lo que quiere decir que cada instrucción tarda un ciclo en ejecutarse. Por ultimo el CPU time es de 724 ps.

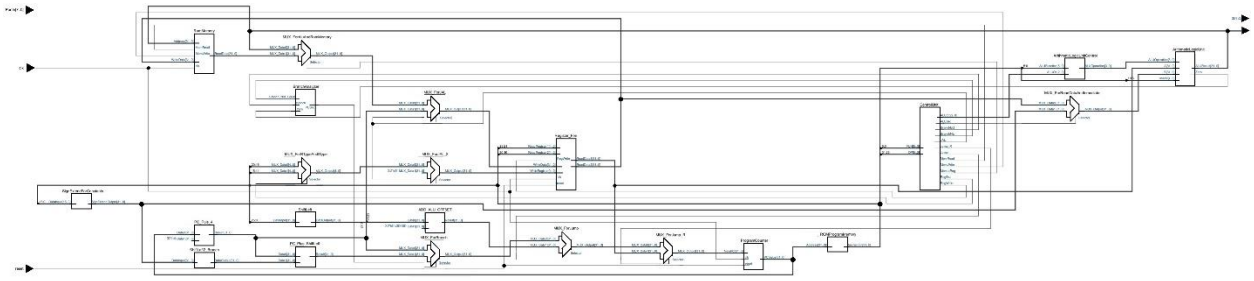
CPU Time:



Clock Rate:

	Fmax	Restricted Fmax	Clock Name	Note
1	42.26 MHz	42.26 MHz	clk	

Esquemático



Para poder adaptar el MIPS previamente diseñado en clase y que fuese apto para ejecutar el algoritmo de las torres de Hanoi de una manera correcta se tuvieron que realizar diversas modificaciones dentro de los módulos ya existentes, así como crear nuevos módulos como se explica a continuación:

Control Unit

En el caso del modulo de control se agregaron mas parámetros locales correspondientes a los códigos de operación de cada instrucción como ADDI, ORI, ANDI, LUI, LW, SW, BEQ, BNE, JUMP y JAL. Además se agregaron mas casos correspondientes con las nuevas instrucciones para encender o apagar los wires de control necesarios para llevar a cabo cada instrucción, adicionalmente se incluyo una sub sentencia dentro del tipo de registro para el caso de JR. Finalmente se agregaron mas wires de control para las nuevas instrucciones dentro de los cuales están incluidos Jal, Jump_R y Jump, por lo que fue necesario crecer el tamaño del registro ControlValues a 14 bits.

ALU

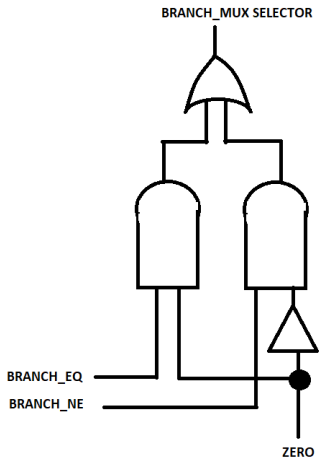
Dentro del modulo de la unidad aritmética lógica se añadieron parámetros locales correspondientes a los resultados del ALUControl relacionados a sus operaciones lógico aritméticas correspondientes, siendo estas SUB, LUI, SLL, y SRL. A su vez se agregaron las respectivas implementaciones para cada una de las instrucciones previamente descritas.

ALUControl

Dentro de este modulo se incluyeron los parámetros locales y casos respectivos para poder mandar al modulo de ALU la respectiva operación a realizar dependiendo de la combinación entre el código de operación y la función. Dentro de las nuevas instrucciones integradas se encuentran NOR, SUBB, SLL, SRL, JR, ANDI, LUI, LW, BE y SW.

BranchAnalizer

Este modulo fue creado de tal manera que permitiera hacer uso de las instrucciones de BEQ y BNE. Para lo cual se utilizo el siguiente diagrama combinacional para poder activar el multiplexor dedicado al Branch de acuerdo al valor de cero obtenido por la ALU y a los cables de control BNE y BEQ salidos de la unidad de control.



Multiplexores

Para la adición del resto de instrucciones necesarias para la practica se agregaron diversos multiplexores para controlar el flujo de datos en base a diversos wires de control. Los multiplexores que se agregaron fueron: un multiplexor dedicado seleccionar el siguiente valor del PC de acuerdo al selector por parte del BranchAnalizer, se agregaron dos multiplexores para manejar la instrucción de jump and link dentro de los cuales se asigna el registro ra como destino y el PC al cual regresar como dato. Por ultimo se agregaron los multiplexores respectivos para el funcionamiento de las instrucciones Jump y Jump register.

PC_Plus_ShiftLeft

Se agregó este modulo para añadir al contador del programa actual el desplazamiento necesario para brincar a la etiqueta establecida dentro de las instrucciones BEQ y BNE

DataMemory

Dentro de este modulo se modifico el parámetro de la profundidad de memoria a un valor de 256 de acuerdo con lo establecido en el enunciado de la practica para poder ejecutar el programa de las torres de Hanoi

ShiftLeft2_Branch

Se agregó este modulo de shift left para poder llevar a cabo la ejecución de las instrucciones BNE y BEQ

RegisterFile

Dentro de este modulo se modifico el valor del parámetro de inicio para el registro de stack pointer, el cual se agrego dentro de los módulos de registro para que cada vez que se llevase a cabo un reinicio el valor volviera a ser el establecido, siendo este 30'hdc

Simulación de instrucciones

A continuación se muestran las ejecuciones de prueba en model sim para las instrucciones beq, bne, lw, sw, j y jr junto con el código utilizado en MARS.

En el siguiente segmento de código se demuestra el funcionamiento de la función beq junto con su simulación dentro de la ilustración 1. Para este escenario se cargaron dos valores diferentes en los registros a comparar para que beq no saltase, después se hicieron ambos registros iguales por lo que al ejecutarse la función beq terminaría saltando a la etiqueta deseada.

```
.text
      addi $t0,$zero,1 #cargamos 1 a t0
      addi $t1,$zero,2 #cargamos 2 a t1
       beq $t0,$t1,label2 #si t0 y t1 son iguales saltamos a label2
       addi $t0,$t0,1    #sumamos 1 a t0
label2:
       beq $t0,$t1,exit #saltamos si t1 y t0 son iguales
       #relleno de instrucciones para probar el beq
       addi $t0,$zero,3
       addi $t1,$zero,3
exit:
      j exit
```

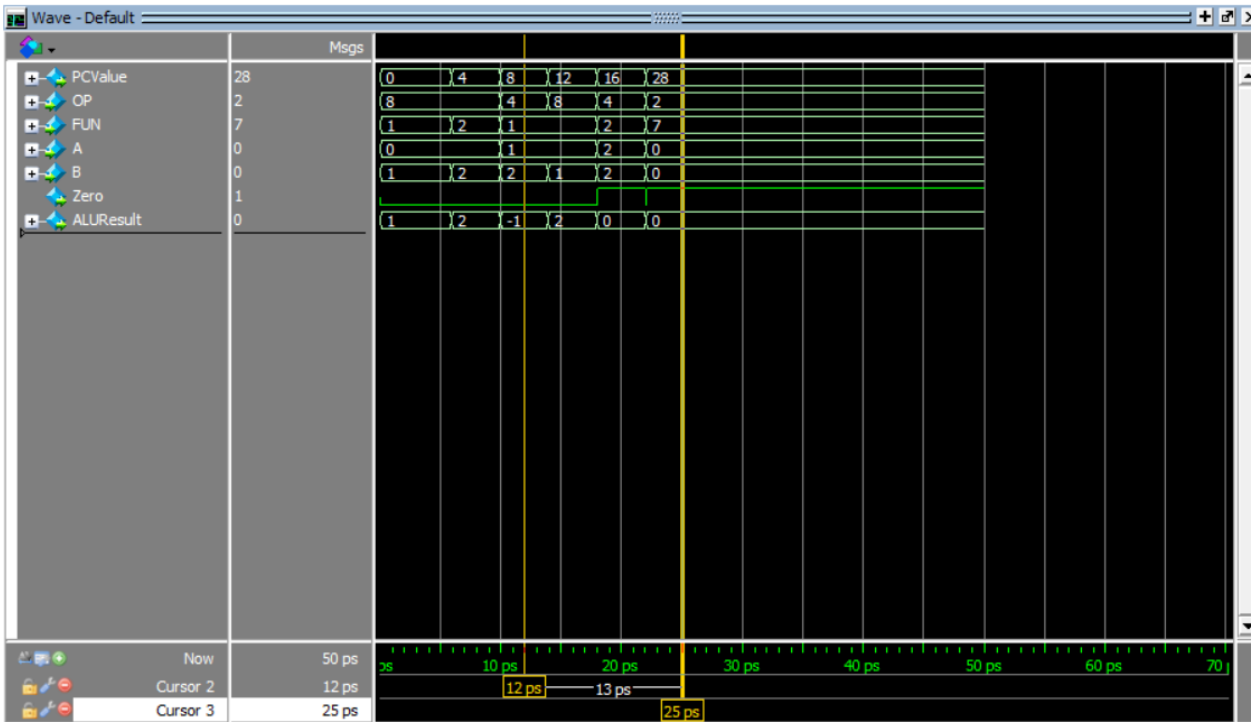


Ilustración 1 Demostracion de BEQ en un escenario donde funciona y uno donde no

Dentro del siguiente segmento de código junto con su simulación dentro de la ilustración 2 se demuestra el funcionamiento de la función bne la cual tiene un parecido con beq pero de forma negada, por lo que se optó por utilizar el mismo código de prueba que en el caso de beq aunque con pequeñas modificaciones para que funcionase, como lo fueron el asignar valores iguales a los registros desde un inicio para despues cambiarlos y que se hicieran diferentes.

```
.text
       addi $t0,$zero,2 #cargamos 2 a t0
       addi $t1,$zero,2 #cargamos 2 a t1
        bne $t0,$t1,label2 #si t0 y t1 no son iguales saltamos a label2
        addi $t0,$t0,1     #sumamos 1 a t0
label2:
        bne $t0,$t1,exit #saltamos si t1 y t0 no son iguales
        #relleno de instrucciones para probar el bne
        addi $t0,$zero,3
        addi $t1,$zero,3
exit:
       j exit
```

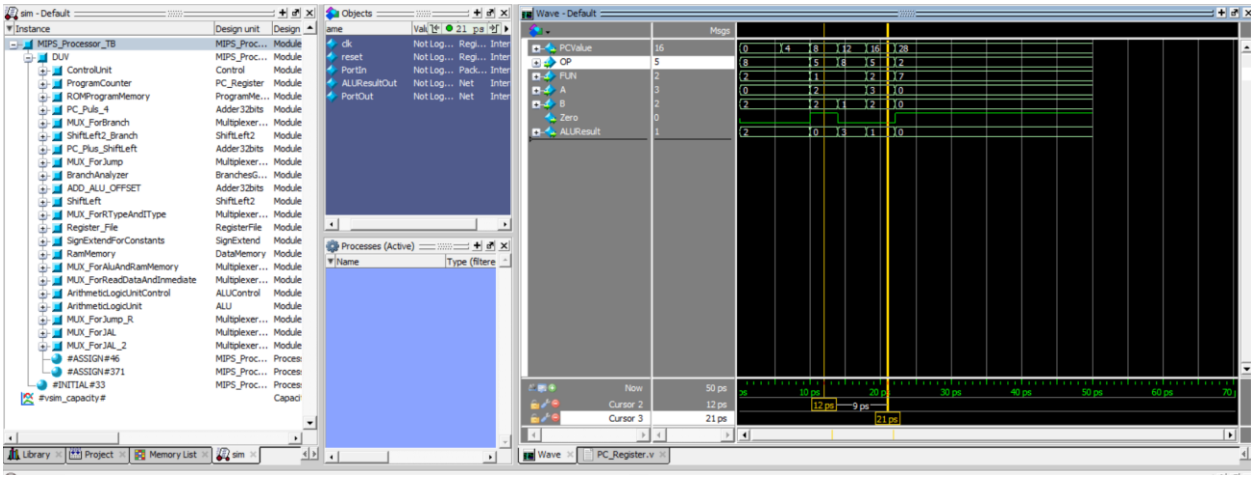


Ilustración 2 Demostracion para bne

En el siguiente segmento de código se muestra la demostración del funcionamiento de las funciones load Word y store Word junto con su ejecución en modelsim incluida en la ilustración 3. Para este escenario se escribió un 2 en el registro t0 para despues ser cargado al stack mediante sw y despues ser traído desde esa misma localidad de memoria hacia el registro t1 mediante lw.

```
.text
        addi $t0,$zero,2 #cargamos 2 a t0
        sw $t0,0($sp)          #guardamos el valor de t0 dentro de la localidad apuntada por el stack pointer
        lw $t1,0($sp)          #cargamos el valor dentro de la localidad apuntada por el stack pointer a t1
exit:
        j exit
```

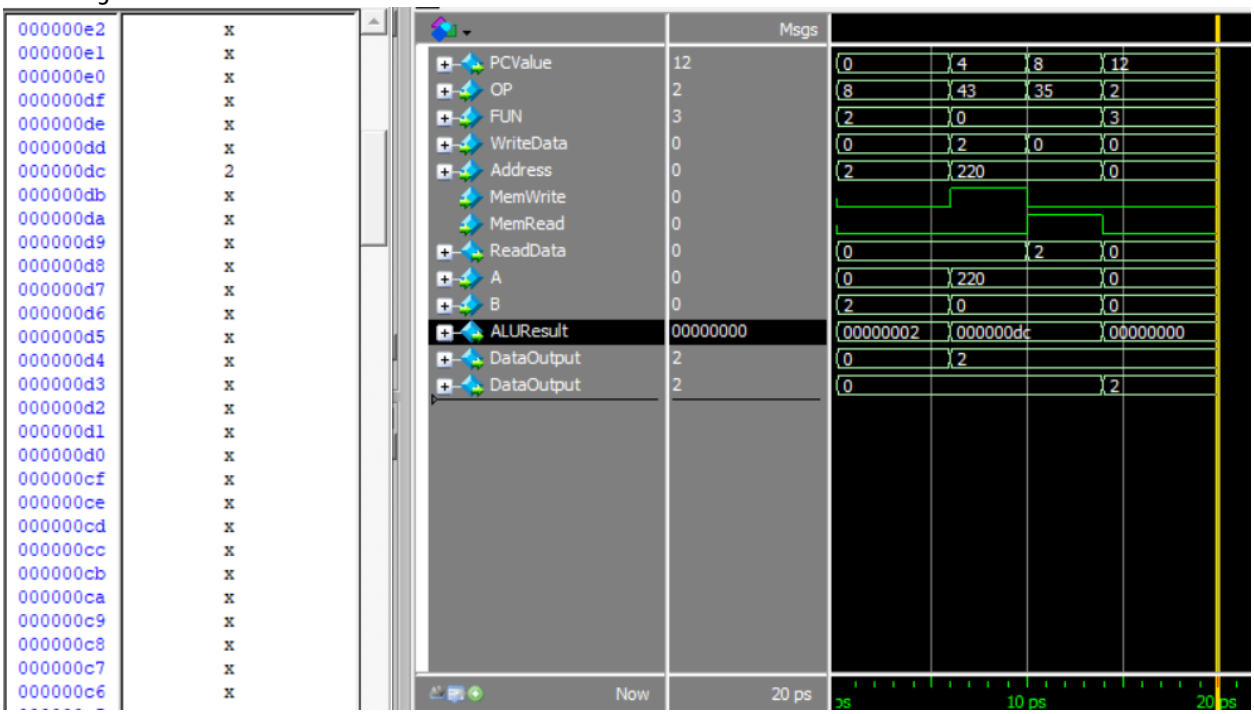


Ilustración 3 Demostracion para sw y lw

En el siguiente segmento de código y su simulación dentro de la ilustración 4 se muestra la ejecución de las funciones jal, j y jr. Dentro de los valores de la simulación se puede observar como el PC no incrementa constantemente, esto debido a los saltos dados en el código, probando así su funcionamiento.

```
.text
       jal parte2
       j exit
parte2:
       jr $ra
exit:
       j exit
```

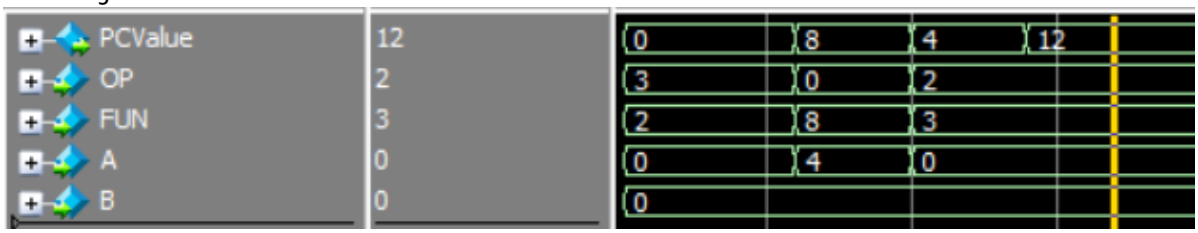


Ilustración 4 Simulacion para j, jal, y jr

Codigo para torres de Hanoi

```
#Alejandro Gudiño
#Ethandrake Castillo
.text
main:
    addi $s0, $s0, 0x000000df #Cargar valor de memoria a registros A, origen
    addi $s1, $s1, 0x000000eb #Cargar valor de memoria a registro B, auxiliar
    addi $s2, $s2, 0x000000f7 #Cargar valor de memoria a registro C, final
    addi $s3, $s3, 3 #Guardar numero de discos

    #Meter los valores en la torre A
    add $t0, $t0, $s0 #Cargar dirección A para manipular
for:    sw $s3, ($t0) #Poner el valor n en torre A
    addi $t0, $t0, 4 #Aumentar un byte de memoria a A
    addi $s3, $s3, -1 #Reducir n
    bne $s3, $zero, for #Regresar a for hasta que n = 0

    addi $s3, $s3, 3 #Guardar número de discos
    jal hanoi
    j exit

hanoi:
        addi $sp, $sp, -20 #Espacio para guardar los 5 registros
    sw $ra, 0($sp) #Guardar RA para poder regresar
    sw $s0, 4($sp) #Guardar dirección de A
    sw $s1, 8($sp) #Guardar dirección de B
    sw $s2, 12($sp) #Guardar dirección de C
    sw $s3, 16($sp) #Guardar N

    #Revisar caso base
```

```
    beq $s3, 1, if #Revisar si n=1
    j else
if:
    #Si n=1
    sw $s3, ($s2) #Mover disco de origen a destino
    sw $zero, ($s0) #Borrar dato de memoria
    j return

else:
    addi $s3, $s3, -1 #Decrementar en 1 el valor de n
    addi $s0, $s0, 4 #Incrementar un byte a la dirección de origen
    lw $s2, 8($sp) #El destino pasa a ser el auxiliar
    lw $s1, 12($sp) #El auxiliar pasa a ser el destino
    jal hanoi
    addi, $s0, $s0, -4 #Decrementar un byte al origen
    lw $s2, 12($sp) #C vuelve a ser el destino
    lw $t0, ($s0)
    sw $t0, ($s2) #Mover el disco del origen al destino
    sw $zero, ($s0) #Borrar disco de torre A
    addi $s2, $s2, 4 #Aumentar en un byte el destino
    lw $s1, 4($sp) #El origen ahora es el auxiliar
    lw $s0, 8($sp) #El auxiliar ahora es el origen
    jal hanoi
return:
    lw $ra, 0($sp) #Sacar valor de RA del stack
    lw $s0, 4($sp) #Sacar direccion de A del stack
    lw $s1, 8($sp) #Sacar direccion de B del stack
    lw $s2, 12($sp) #Sacar direccion de C del stack
    lw $s3, 16($sp) #Sacar valor de n de la pila
    addi $sp, $sp, 20 #Regresar el stacka su valor normal
    jr $ra  #Fin de la funcion
exit:
```

Diagrama de Flujo para el Algoritmo de las Torres de Hanoi

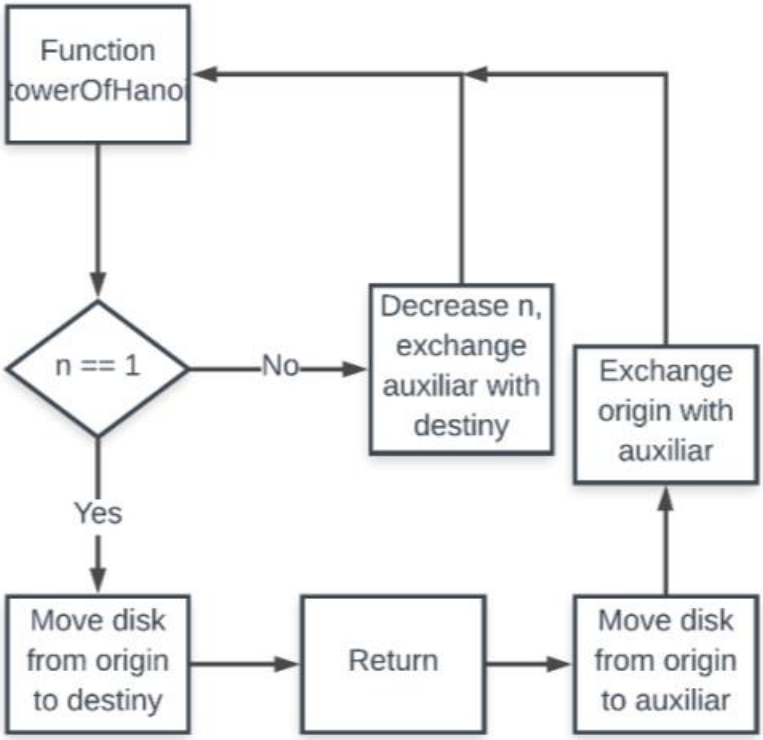


Ilustración 5 Diagrama de Flujo Torres de Hanoi

Ejecución de las torres de Hanoi con 3 discos

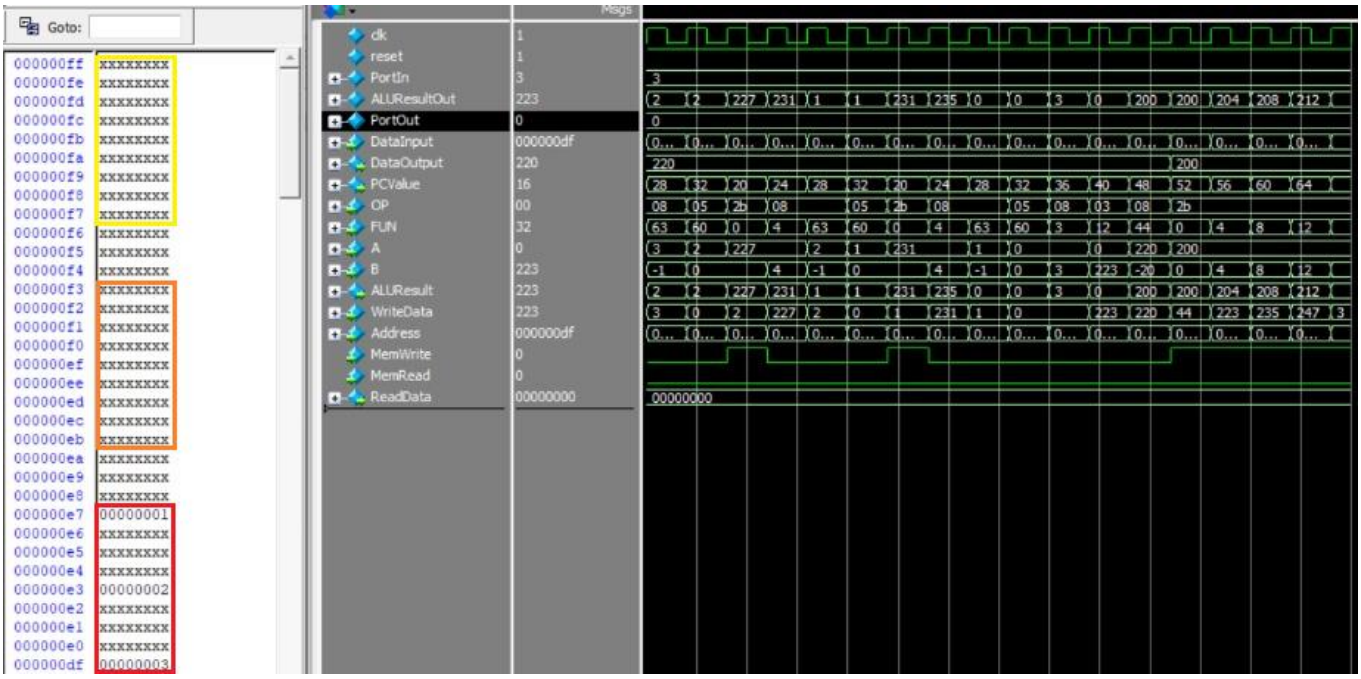


Ilustración 6 Ejecución de las torres de Hanoi con 3 discos / primer movimiento

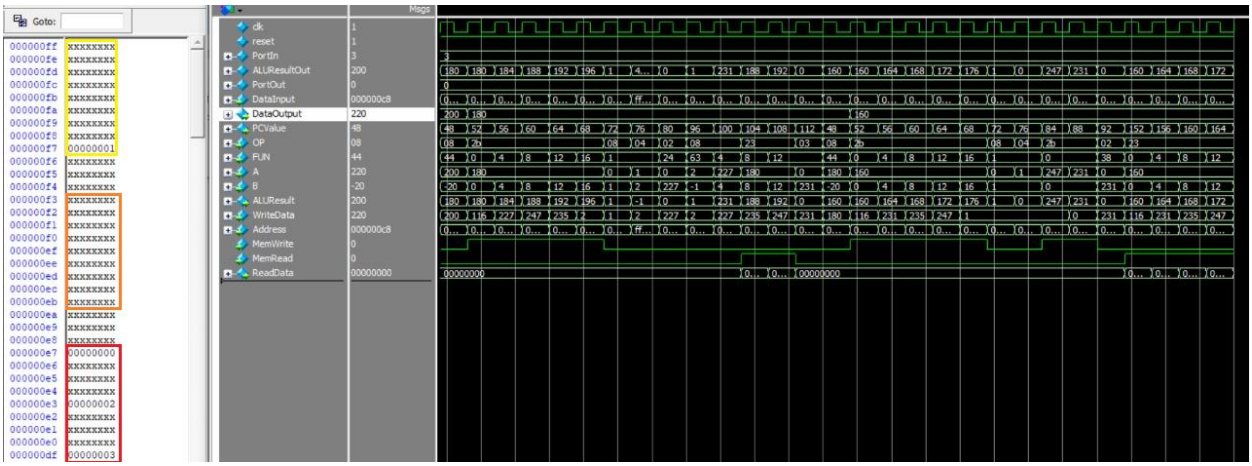


Ilustración 7 Ejecución de las torres de Hanoi con 3 discos / segundo movimiento

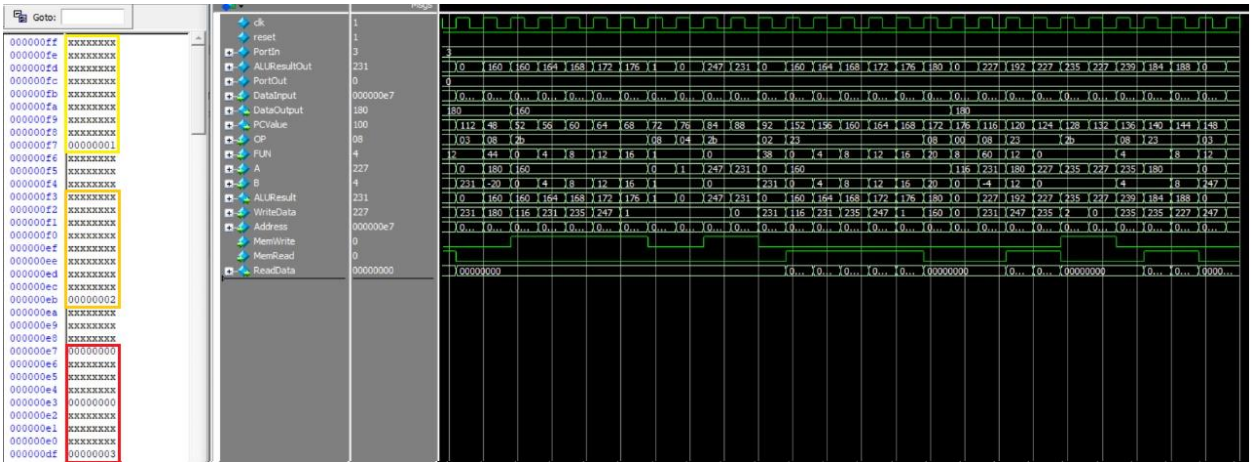


Ilustración 8 Ejecución de las torres de Hanoi con 3 discos / tercer movimiento

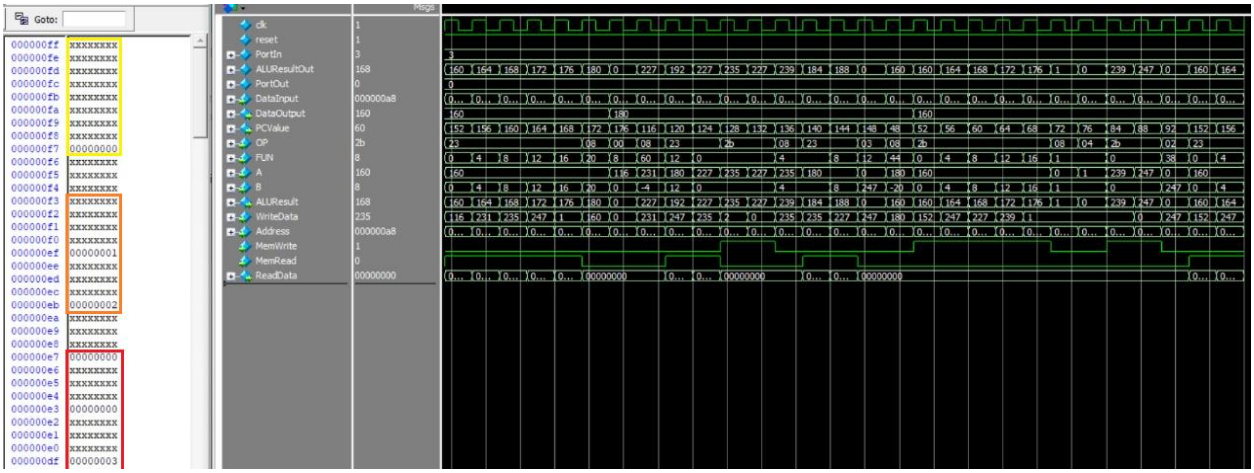


Ilustración 9 Ejecución de las torres de Hanoi con 3 discos / cuarto movimiento

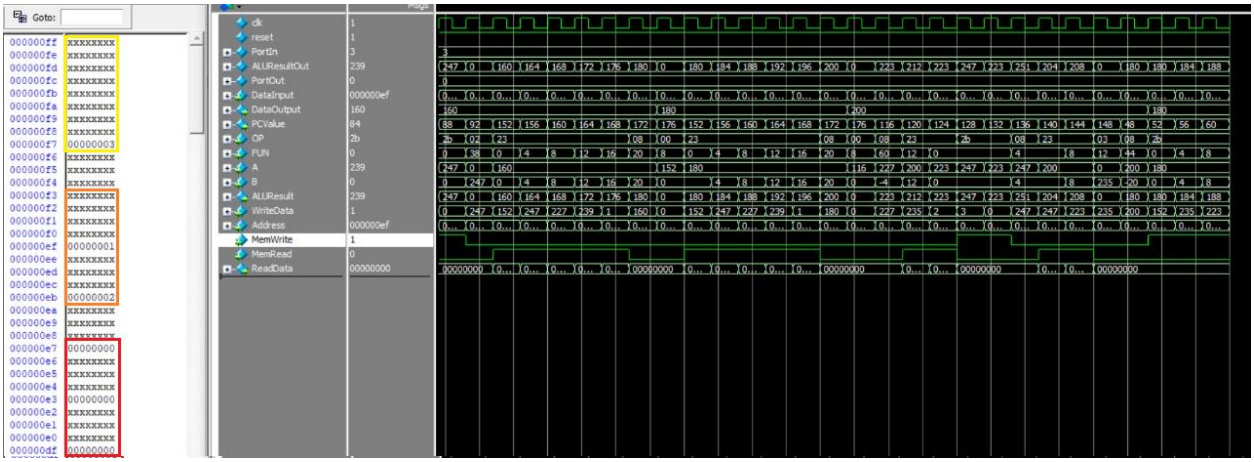


Ilustración 10 Ejecución de las torres de Hanoi con 3 discos / quinto movimiento

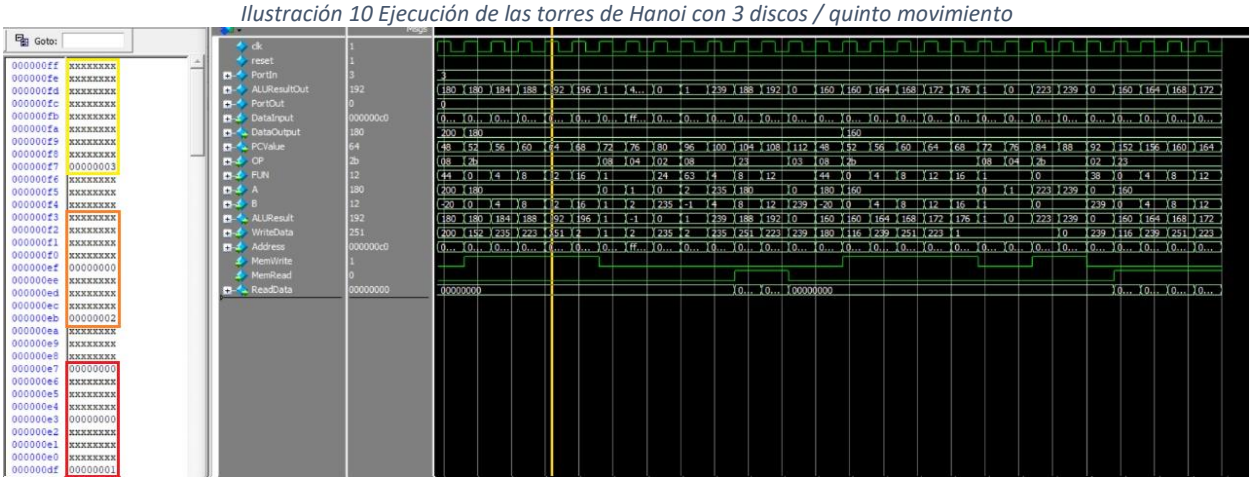


Ilustración 11 Ejecución de las torres de Hanoi con 3 discos / sexto movimiento

Ilustración 12 Ejecución de las torres de Hanoi con 3 discos / séptimo movimiento

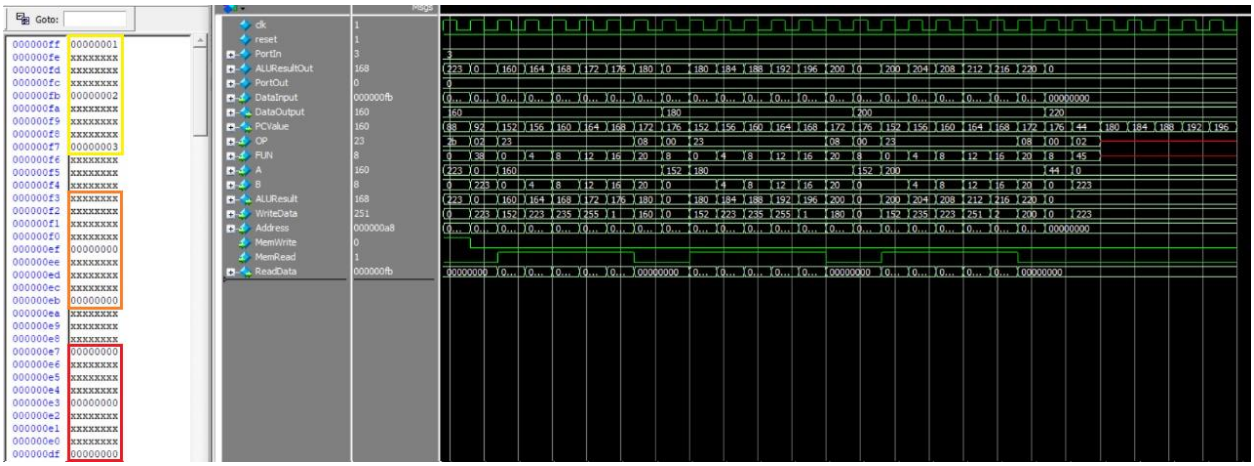


Ilustración 13 Ejecución de las torres de Hanoi con 3 discos / octavo movimiento